Today

- ttcp
- select
- daemons
- syslog

ascii to IP address

```
char host[MAXLTH];  /*hostname or w.x.y.z */
...
sin.sin_addr.s_addr = aton(host);
...

unsigned int aton(char *s)
{
       unsigned int addr;
       struct hostent *h;

       addr = inet_addr(s);
       if (addr != -1) return (addr);
       h = gethostbyname(s);
       if (h == NULL) err_sys("bad host name/addr");
       bcopy(h->h_addr, (char *)&addr, h->h_length);
       return (addr);
 }
```


ttcp

- TCP and UDP performance tester
- source/sink test
- debug and setsockopt() options
- widely used
- requires accounts on both systems
- client and server
- alternates: **spray** or **netperf** or **sock**

dunigan/cs494-unp/ttcp.c and ttcp.1 and web links

ttcp name/ip resolve

```
...
      if (atoi(host) > 0 )  {
         /* Numeric */
         sinhim.sin_family = AF_INET;
         sinhim.sin_addr.s_addr = inet_addr(host);
      } else {
         if ((addr=gethostbyname(host)) == NULL)
                 err("bad hostname");
         sinhim.sin_family = addr->h_addrtype;
         bcopy(addr->h_addr,(char*)&addr_tmp, addr->h_length);
         sinhim.sin_addr.s_addr = addr_tmp;
      }
      sinhim.sin_port = htons(port);
...
```

ttcp runs

```
duncan> ttcp -r -s
ttcp-r: nbuf=1024, buflen=1024, port=2000
ttcp-r: socket
ttcp-r: accept
ttcp-r: 0.1user 0.5sys 0:19real 3% 0i+31d 19maxrss 0+3pf 1862+57csw
ttcp-r: 1048576 bytes processed
ttcp-r:      0.69 CPU sec  =   1484.06 KB/cpu sec,    11872.5 Kbits/cpu sec
ttcp-r:   19.0521 real sec =   53.7474 KB/real sec,   429.979 Kbits/sec
----
menkar% ttcp -t -s duncan.cs.utk.edu
ttcp-t: nbuf=1024, buflen=1024, port=2000
ttcp-t: socket
ttcp-t: connect
ttcp-t: 0.0user 2.7sys 0:19real 14% 0i+18d 18maxrss 0+0pf 941+914csw
ttcp-t: 1048576 bytes in 2.780000 CPU seconds = 368.345324 KB/cpu sec
ttcp-t: 1048576 bytes in 18.999998 real seconds = 53.894743 KB/sec
```


ttcp runs

```
duncan> ttcp -r -s -u
ttcp-r: nbuf=1024, buflen=1024, port=2000
ttcp-r: socket
ttcp-r: 0.0user 0.0sys 0:01real 4% 0i+47d 23maxrss 0+0pf 270+5csw
ttcp-r: 276480 bytes processed
ttcp-r:      0.09 CPU sec  =      3000 KB/cpu sec,      24000 Kbits/cpu sec
ttcp-r:   1.93405 real sec =   139.603 KB/real sec,   1116.83 Kbits/sec
-------------
menkar% ttcp -t -s -u duncan.cs.utk.edu
ttcp-t: nbuf=1024, buflen=1024, port=2000
ttcp-t: socket
ttcp-t: 0.0user 1.7sys 0:01real 100% 0i+17d 17maxrss 0+0pf 0+18csw
ttcp-t: 1048576 bytes in 1.840000 CPU seconds = 556.521739 KB/cpu sec
ttcp-t: 1048576 bytes in 1.840000 real seconds = 556.521739 KB/sec
```


ttcp runs

SNDBUF RCVBUF

```
msr% ttcp -r -s -b 50000
ttcp-r: nbuf=1024, buflen=1024, port=2000
ttcp-r: socket
ttcp-r: accept
ttcp-r: 0.0user 0.0sys 0:00real 9% 0i+53d 26maxrss 0+0pf 118+0csw
ttcp-r: 1048576 bytes processed
ttcp-r:      0.03 CPU sec  =   34133.3 KB/cpu sec,     273067 Kbits/cpu sec
ttcp-r:  0.313238 real sec =   3269.08 KB/real sec,   26152.6 Kbits/sec
-------------
icarus% ttcp -t -s -b 50000 msra
ttcp-t: nbuf=1024, buflen=1024, port=2000
ttcp-t: socket
ttcp-t: connect
ttcp-t: 0.0user 0.2sys 0:00real 103% 0i+68d 34maxrss 0+0pf 0+4csw
ttcp-t: 1048576 bytes processed
ttcp-t:       0.3 CPU sec  =   3413.33 KB/cpu sec,    27306.7 Kbits/cpu sec
ttcp-t:  0.292385 real sec =   3502.23 KB/real sec,   28017.9 Kbits/sec
```


ttcp.c

usage (varies with version)

```
Usage: ttcp -t [-options] host <in
   -l ##  length of bufs written to network (default 1024)
   -s source a pattern to network
   -n ##  number of bufs written to network (-s only, default 1024)
   -p ##  port number  (default 2000)
   -u use UDP instead of TCP
   -b ### set socket buffer size
   -d SO_DEBUG
   -A align start of buffers
   -D don't buffer writes TCP_NODELAY
```

concurrency

three ways to avoid blocking

- fork()
- threads
- select()
- nonblocking IO (poll/EWOULDBLOCK)
- signal-driven IO
- asynchronous IO

blocking calls: read, recvfrom, accept, write (buffs full)

select()

```
#include <sys/types.h>
#include <sys/time.h>

int select (width, readfds, writefds, exceptfds, timeout)
int width;
fd_set *readfds, *writefds, *exceptfds;
struct timeval *timeout;

FD_SET (fd, &fdset)
FD_CLR (fd, &fdset)
FD_ISSET (fd, &fdset)
FD_ZERO (&fdset)
int fd;
fd_set fdset;
```

- select() returns count of ready fd's
- width is max number of fd's, which can be largest fd open plus one
- if **timeout** is NULL, block til ready or signal (EINTR)
- if **timeout** is 0, poll
- if **timeout** is non-zero, then if timer expires, select() returns 0

the fd masks

- bit masks indicating which fd's to test
- masks are modified after call (value-result args)
- NULL may be supplied, if not interested
- **readfds** descriptors ready for reading, or ready for **accept()** if **listen()** issued
- **writefds** descriptors ready for writing
- **exceptfds** arrival of urgent data

doin' it

- FD_ZERO the master mask
- FD_SET the relevant fd's in the relevant mask (usually readfds)
- copy master to working mask
- issue select with working mask and hi-fd count
- check return value (it's interruptible, EINTR)
- find out which fd's are set (FD_ISSET) and do appropriate I/O operation (read/write, recvfrom, etc.)
- if an error or EOF occurs on an fd operation (read/write), need to FD_CLR fd in master mask (assuming your watching more than one fd)
- go back to copy master step ...

cat select

```
/* select.c */
#include <stdio.h>
#include <sys/types.h>
main()
{
   int n,nfds =1;
   fd_set rdfds, rdfds_master;
   char buf[1024];

   FD_ZERO(&rdfds_master);
   FD_SET(0,&rdfds_master);

   for(;;){
      rdfds = rdfds_master; /* reset */
      if ( select(nfds, &rdfds, NULL,NULL,NULL) <0)
       err_sys("select");
      if (FD_ISSET(0,&rdfds)){
         n=read(0,buf,sizeof(buf));
         if (n<0) err_sys("read");
         if (n == 0) break;
         write(1,buf,n);
      }
   }
}
```


select with timeout

```
main()
{
   int n,nfds =1;
   fd_set rdfds, rdfds_master;
   static struct timeval timeout = {3,0};
   char buf[1024];

   FD_ZERO(&rdfds_master);
   FD_SET(0,&rdfds_master);

   for(;;){
      rdfds = rdfds_master; /* reset */
      if ( (n=select(nfds, &rdfds, NULL,NULL,&timeout))<0)
        err_sys("select");
      if (n == 0) {
         printf("\ntimed out! bye\n");
         exit(1);
      }
      if (FD_ISSET(0,&rdfds)){
         n=read(0,buf,sizeof(buf));
         if (n<0) err_sys("read");
         if (n == 0) break;
         write(1,buf,n);
      }
   }
   exit(0);
}
```


delay.c

```
main()
{
   double t,seconds();
   t= seconds();
   delay(3,700000);
   t = seconds()-t;
   printf("%f\n",t);
}
delay(s,us)
{
   struct timeval tv;

   tv.tv_sec = s;
   tv.tv_usec = us;
   (void)select( 0, NULL, NULL, NULL, &tv );
}

double
seconds()
{
   struct timeval ru;
   gettimeofday(&ru, (struct timezone *)0);
   return(ru.tv_sec + ((double)ru.tv_usec)/1000000);
}
```


gethostbyname

```
...
  s = socket(AF_INET, SOCK_DGRAM, 0);
  if (sendto(s, buf, buflen, 0, &_res.nsaddr_list[ns],
   sizeof(struct sockaddr)) != buflen) {
     if (_res.options & RES_DEBUG)
      perror("sendto");
      continue;
   }
   timeout.tv_sec = (_res.retrans << (_res.retry - retry))
     / _res.nscount;
   if (timeout.tv_sec <= 0)timeout.tv_sec = 1;
   timeout.tv_usec = 0;

wait:
   FD_ZERO(&dsmask);
   FD_SET(s, &dsmask);
   n = select(s+1, &dsmask, (fd_set *)NULL,
       (fd_set *)NULL, &timeout);
   if (n == 0) {
       if (_res.options & RES_DEBUG)
          printf("timeout\n");
      continue
   }
   if ((resplen = recv(s, answer, anslen, 0)) <= 0) {
   ...
```

strcliselect01.c fig 6.9

```
{
        int    maxfdp1;
        fd_set  rset;
        char   sendline[MAXLINE], recvline[MAXLINE];

        FD_ZERO(&rset);
        for ( ; ; ) {
                FD_SET(fileno(fp), &rset);
                FD_SET(sockfd, &rset);
                maxfdp1 = max(fileno(fp), sockfd) + 1;
                Select(maxfdp1, &rset, NULL, NULL, NULL);
```



```
                if (FD_ISSET(sockfd, &rset)) {  /* socket is readable */
                        if (Readline(sockfd, recvline, MAXLINE) == 0)
                                err_quit("str_cli: server terminated prematurely");
                        Fputs(recvline, stdout);
                }

                if (FD_ISSET(fileno(fp), &rset)) {  /* input is readable */
                        if (Fgets(sendline, MAXLINE, fp) == NULL)
                                return;         /* all done */
                        Writen(sockfd, sendline, strlen(sendline));
                    }
         }
        }
```


daemons

- detached processes (server)
- no controlling tty
- started at boot time (*/etc/rc**)
- run forever
- await service request
- spawn subprocesses to handle requests

how to become a daemon

- fork(), let parent exit
- disassociate from process group
- disassociate from controlling tty
- close any files
- chdir to / and set umask
- do something useful

varies per OS

```
/* mydaemon.c */
#include <stdio.h>
#include <sys/file.h>
#include <sys/ioctl.h>
main(){
   register int   childpid, fd;
   /* fork and let parent exit */
   if ( (childpid = fork()) < 0){
      perror("can't fork first child");
      exit(1);
   } else if (childpid > 0){
      printf("daemon pid is %d\n",childpid);
      exit(0); /*parent leaves*/
   }
   /* Disassociate from controlling terminal
    * and process group */
   if (setpgrp(0, getpid()) == -1){
      perror("can't change process group");
      exit(1);
   }
   if ( (fd = open("/dev/tty", O_RDWR)) >= 0) {
      /* lose controlling tty */
      ioctl(fd, TIOCNOTTY, (char *)NULL);
      close(fd);
   }
   for (fd = 0; fd < getdtablesize(); fd++)
      close(fd);  /* close any open files */
   chdir("/");  /* avoid mounted filesystems */
   umask(0);
     pause();  /* do daemon things .... */
}
```


Debuggable daemons

- Don't detach

  command line option
  don't fork
  keep std-files open
  if (debug) ...
- use **syslog()**
- use SIGUSR1/2

SIG debugging

```
...
signal(SIGUSR1,db_dump);
signal(SIGUSR2,db_reload);
...
db_dump()
{
   /* dump daemon's data base to /tmp/....
   ....
}
```

user types **kill -USR1 324**

syslog

- error logging
- handy for daemons or you
- system calls **openlog(), syslog()**
- logging daemon *syslogd*

  uses AF_UNIX socket
  configuration */etc/syslog.conf*

also can use *logger* command

/etc/syslog.conf

```
# syslog configuration file.
#  facility.level(priority)     where to send messages
*.err;kern.debug;auth.notice;user.none    /dev/console
*.err;kern.debug;mail.crit;user.none /var/log/messages

mail.debug;*.info  ifdef(`LOGHOST', /var/log/syslog, @loghost)
*.alert;kern.err;daemon.err;auth.none  operator
*.alert;kern.err;daemon.err;auth.none  @hasbro
kern.alert @ft-knox
```


syslog functions

#include <syslog.h>

openlog(char *ident, logopt, facility)

syslog(priority, char *message, ...)

```
#include <syslog.h>
main(){
  openlog("tomtest",LOG_PID,LOG_MAIL);
  syslog(LOG_INFO,"sys log test");
  syslog(LOG_AUTH|LOG_NOTICE,"sys log test auth/notice");
}

and what appears in log

Sep 22 07:51:30 thistle tomtest[9729]: sys log test
Sep 22 07:51:30 thistle tomtest[9729]: sys log test auth/notice
```

openlog() is optional

syslog in libc

- set up UDP socket (AF_UNIX)
- sendto() to host's syslogd

```
char     *logname = "/dev/log";
...
SyslogAddr.sa_family = AF_UNIX;
strncpy(SyslogAddr.sa_data, logname,
  sizeof SyslogAddr.sa_data);
LogFile = socket(AF_UNIX, SOCK_DGRAM, 0);
... construct message
sendto(LogFile, outline, c, 0, &SyslogAddr,
  sizeof SyslogAddr);
```

```
datagram: <fac|pri> time tag [pid]: message
```


syslogd

- example of a daemon
- started at system startup
- reads /etc/syslog.conf
- awaits input from local UDP, remote UDP (514), or /dev/klogd
- parses message and sends to appropriate destinations
- uses *sendto()* for remote (@) entries

see dunigan/cs494-unp/syslogd.c

syslogd.c

daemonize

```
...
 if (!Debug) {
         int i;
         if (fork()) exit(0);
         for (i = 0; i < 10; i++)
                 (void) close(i);
         (void) open("/", 0);
         (void) dup2(0, 1);
         (void) dup2(0, 2);
         i = open("/dev/tty", O_RDWR);  /* untty */
         if (i >= 0) {
            ioctl(i, (int) TIOCNOTTY, (char *)0);
            close(i);
         }
 } else
...
```


syslogd.c

signals

```
  signal(SIGTERM, die);
  signal(SIGINT, Debug ? die : SIG_IGN);
  signal(SIGQUIT, Debug ? die : SIG_IGN);
  signal(SIGCHLD, reapchild);
  ...
void reapchild()
{
    union wait status;

    while (wait3(&status, WNOHANG,
        (struct rusage *) NULL) > 0) ;
}

void die(sig)
{
    if (sig) {
         dprintf("syslogd: going down on signal %d\n", sig);
          ..
    }
    unlink(LogName);
    exit(0);
}
```


syslogd.c

input setup

```
sunx.sun_family = AF_UNIX;
strncpy(sunx.sun_path, LogName, sizeof sunx.sun_path);
funix = socket(AF_UNIX, SOCK_DGRAM, 0);
bind(funix, (struct sockaddr *) &sunx,lth);

finet = socket(AF_INET, SOCK_DGRAM, 0);
sp = getservbyname("syslog", "udp");
sin.sin_family = AF_INET;
sin.sin_port = LogPort = sp->s_port;
bind(finet, (struct sockaddr *)&sin, sizeof(sin));

fklog = open("/dev/klog", O_RDONLY);
```


syslogd.c

main loop

```
  init();  /* read config file */
  signal(SIGHUP, init);
  for (;;) {
     int nfds, readfds = FDMASK(funix) | inetm | klogm;

     nfds = select(20, (fd_set *) &readfds,NULL,NULL,NULL);
     if (nfds == 0) continue;
     if (nfds < 0) {
       if (errno != EINTR) logerror("select");
       continue;
      }
     if (readfds & klogm) {
        i = read(fklog, line, sizeof(line) - 1);
        printsys(line);
     }
     if (readfds & FDMASK(funix)) {
        i = recvfrom(funix, line, MAXLINE, 0,
           (struct sockaddr *) &fromunix, &len);
        printline(LocalHostName, line, USER_ONLY);
     }
     if (readfds & inetm) {
        len = sizeof frominet;
        i = recvfrom(finet, line, MAXLINE, 0,
           (struct sockaddr *)&frominet, &len);
        printline(cvthname(&frominet), line, ALL);
     }
  }
```

next time

inetd