Today

- socket options
- UDP
- signals

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();
```

or if in separate file, remember to *double seconds();*

tools

commands

- nslookup
- netstat -a
- strace/truss/trace

library

- readn()/writen()
- seconds()
- sys_err()
- develop wrappers

understand possible failures

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);
}
```


TCP

Transmission Control Protocol (RFC793)

- connection-oriented
- 16-bit port
- reliable
- timers, checksums, sequence numbers
- flow control (end-to-end), 16-bit window

UDP

User Datagram Protocol (RFC768)

- connectionless (datagram)
- stateless, light-weight
- 16-bit port and length
- unreliable (lost, damaged, duplicated, delayed, out of sequence)
- optional checksum
- needed for multicast/broadcast
- tftp,ntp,snmp, audio/video, nfs, dns, rpc

UDP calls

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

  int sendto(int s, const void *msg, int len,  unsigned  int
  flags, const struct sockaddr *to, int tolen);

  int recvfrom(int s, void *buf, int len, unsigned int flags
  struct sockaddr *from, int *fromlen);
```

- can send and receive 0 bytes (no EOF)
- flags usually 0
- *bind()* is optional for client
- failures: too big, buffs full, interrupted

udpcli.c

```
main(argc, argv)
int   argc;
char  *argv[];
{
   int         sockfd;
   struct sockaddr_in   cli_addr, serv_addr;
   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_UDP_PORT);
   if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
      err_dump("client: can't open datagram socket");
   bzero((char *) &cli_addr, sizeof(cli_addr)); /* zero out */
   cli_addr.sin_family      = AF_INET;
   cli_addr.sin_addr.s_addr = htonl(INADDR_ANY);
   cli_addr.sin_port        = htons(0);
   if (bind(sockfd, (struct sockaddr *) &cli_addr,
     sizeof(cli_addr)) < 0)
      err_dump("client: can't bind local address");
   dg_cli(stdin, sockfd, (struct sockaddr *) &serv_addr,
 sizeof(serv_addr));
   close(sockfd);
   exit(0);
}
```


dgcli.c +

```
dg_cli(fp, sockfd, pserv_addr, servlen)
FILE     *fp;
int      sockfd;
struct sockaddr   *pserv_addr;   /* ptr to sockaddr_XX */
int servlen;   /* actual sizeof(*pserv_addr) */
{
   int n,fromlen;
   struct sockaddr from;
   char  sendline[MAXLINE], recvline[MAXLINE + 1];

   while (fgets(sendline, MAXLINE, fp) != NULL) {
      n = strlen(sendline);
      if (sendto(sockfd, sendline, n, 0, pserv_addr, servlen) != n)
         err_dump("dg_cli: sendto error on socket");
      fromlen = sizeof(from);  /* +++ */
      n = recvfrom(sockfd, recvline, MAXLINE, 0,&from,&fromlen);
      if (n < 0)
         err_dump("dg_cli: recvfrom error");
      recvline[n] = 0;  /* null terminate */
      fputs(recvline, stdout);
   }

   if (ferror(fp))
      err_dump("dg_cli: error reading file");
}
```


udpserv.c

```
main(argc, argv)
int   argc;
char  *argv[];
{
   int         sockfd;
   struct sockaddr_in   serv_addr, cli_addr;
   if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
      err_dump("server: can't open datagram 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_UDP_PORT);

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

   dg_echo(sockfd,(struct sockaddr *)&cli_addr,sizeof(cli_addr));
}
```

dgecho.c

```
dg_echo(sockfd, pcli_addr, maxclilen)
int      sockfd;
struct sockaddr   *pcli_addr; /*  sockaddr_XX structure */
int      maxclilen;  /* sizeof(*pcli_addr) */
{
   int   n, clilen;
   char  mesg[MAXMESG];

   for ( ; ; ) {
      clilen = maxclilen;
      n = recvfrom(sockfd, mesg, MAXMESG, 0, pcli_addr, &clilen);
      if (n < 0)
         err_dump("dg_echo: recvfrom error");

      if (sendto(sockfd, mesg, n, 0, pcli_addr, clilen) != n)
         err_dump("dg_echo: sendto error");
   }
}
```


UDP and connect

- If your UDP client is only going to one server, then connect() will inform you on first write() if service is not available.
- use of connect() permits you to use read/write but have to modify use of send/recv/sendto/recvfrom

```
...
   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sin_family = server->h_addrtype;
   bcopy(server->h_addr, (caddr_t)&serv_addr.sin_addr,
      server->h_length);
   serv_addr.sin_port        = htons(port);
   sockfd = socket(AF_INET, SOCK_DGRAM, 0);
#ifdef CONNECT
   connect(sockfd,&serv_addr,sizeof(serv_addr));
   sendto(sockfd,buff,lths[i],0,NULL,0);
#else
   sendto(sockfd,buff,lths[i],0,&serv_addr,sizeof(serv_addr));
#endif
   ...
```


UDP timeouts

client will worry about lost packets

```
...
void ding(int signo) { timedout=1;}
...
   signal(SIGALRM, ding);
   siginterrupt(SIGALRM, 1);   /* don't restart syscalls */
   sendto(sockfd,buff,lths[i],0,NULL,0);
   timedout=0;
   alarm(SECS);
   if (recvfrom(sockfd,buff,lths[i],0,&from,&fromlen) <0){
     if (timedout) lost++;
     else err_sys("recvfrom");
   }
   alarm(0);
...
```


rdate

```
/*  rdate.c   udp version */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <signal.h>
#define BASE1970  2208988800L /* difference between Unix time and ne
#define SECS 3
int timedout;
void ding(int signo) { timedout=1; }
main (argc,argv)
int argc;
char *argv[];
{
   int i;
   if (argc == 1) {
      printf("usage: rdate <host1> <host2> <host...>\n");
      exit(1);
   }
   signal(SIGALRM,ding);
   siginterrupt(SIGALRM, 1);   /* don't restart syscalls */
   for( i = 1; i < argc; i++ )
      RemoteData(argv[i]);
}
```

```
RemoteData(host)
char *host;
{
   struct   hostent *him;     /* host table entry */
   struct   servent *timeServ;   /* sevice file entry */
   struct   sockaddr_in sin;  /* socket address */
   struct sockaddr from;
   int fromlen = sizeof(from);
   int   fd;         /* network file descriptor */
   long  unixTime;      /* time in Unix format */
   u_char  netTime[4];     /* time in network format */
   int   i;       /* loop variable */
   char  *ctime();

   if ((him = gethostbyname(host)) == NULL) {
      fprintf(stderr, "rdate: Unknown host %s\n", host);
      return(-1);
   }
  if ((timeServ = getservbyname("time","udp")) == NULL) {
      fprintf(stderr, "rdate: time/udp: unknown service\n");
      return(-1);
   }
   if ((fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      perror("rdate");
      return(-1);
   }
```



```
   sin.sin_family = him->h_addrtype;
   bcopy(him->h_addr, (caddr_t)&sin.sin_addr, him->h_length);
   sin.sin_port = timeServ->s_port;
   printf("[%s]\t", him->h_name);
   if (sendto(fd,netTime,0,0,&sin,sizeof(sin))<0){
         perror("rdate send");
         close(fd);
         return(-1);
    }
   /* read in the response */
   /**   for udp  need timeout here and verify size is right size **
   timedout=0;
   alarm(SECS);
   i = recvfrom(fd,netTime,sizeof(netTime),0,&from,&fromlen);
   if (i != sizeof(netTime) || timedout) {
         if (timedout)printf(" no response\n");
           else  perror("rdate recv");
         close(fd);
         return(-1);
   }
   alarm(0);
   close(fd);
   unixTime = ntohl(* (long *) netTime) - BASE1970;
   printf("%s", ctime(&unixTime));
}
```


What if?

things that go bump in the net

1. UDP sendto, and no server process
2. UDP sendto with connect(), and no server process
3. active UDP session, ctrl-c server
4. inactive UDP session, server computer crashes and reboots
5. server tries to bind to port $<$ 1024
6. start 2nd copy of server
7. A sends faster than B can receive (UDP)
8. A sends faster than its interface (ENOBUFS)
9. A sends 10000 byte datagram
10. A sends 500-byte datagram, B recvfrom len is 100
11. A sending to B, can C send to B?
12. fd's/host (TCP vs UDP)

netstat -s may show overruns

increase SO_RCVBUFF may help

Signals

- software interrupts (event notification)
- synchronous

  addressing error

  illegal instruction

  floating pt. exception
- asynchronous

  I/O-network events

  timer interrupts

  process termination

  ctrl-* or kill -SIG
- often non-portable
- used to be unreliable

portability

Beware!

- BSD, Posix, Sys V, Linux
- installed once ?
- blocking calls resumed?
- signal queueing
- based on sigaction()
- watch out for race conditions

Posix semantics

- installed once
- during handling, same signal is blocked
- if multiple signal, only one delivered
- slow system calls (accept/recvfrom/select/read) may not be restarted, *errno == EINTR*
- *connect()* cannot be restarted manually, need *select()*

book's wrapper

```
#include        "unp.h"

Sigfunc *
signal(int signo, Sigfunc *func)
{
        struct sigaction        act, oact;

        act.sa_handler = func;
        sigemptyset(&act.sa_mask);
        act.sa_flags = 0;
        if (signo == SIGALRM) {
#ifdef  SA_INTERRUPT
           act.sa_flags |= SA_INTERRUPT;   /* SunOS 4.x */
#endif
        } else {
#ifdef  SA_RESTART
           act.sa_flags |= SA_RESTART; /* SVR4, 44BSD */
#endif
        }
        if (sigaction(signo, &act, &oact) < 0)
                return(SIG_ERR);
        return(oact.sa_handler);
}
```


Generating signals

- by process itself
- by kernel
- by another process (kill())
- by user (kill, ctrl-*)

Signal calls

void (*signal (int sig, void (*disp)(int)))(int)

- SIG_IGN to ignore, or
- SIG_DFL to revert to default, or
- handler to call when signal occurs
- returns previous setting
- one handler can handle multiple signals
- handler remains installed after signal
- signal held during handling

int sigpause(int sigmask)
int sigblock(int mask)
int sigsetmask(int mask)
int sigmask(int sig)
int kill(int pid, int sig)


```
/* cleanup.c */
#include <signal.h>
void bang(int signo)
{
        printf("done\n");
        exit(0);
}

main()
{
        int i;

        signal(SIGINT,bang);
        signal(SIGTERM,bang);

        for(;;)i++;
}
```



```
/* sigenv.c */
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#define SECS 3
#define MAXLTH 256
int timedout;
...
```



```
void ding(int signo){ timedout = 1; }
main()
{
   int status,ding();
   char *getenv(), var[MAXLTH],*val;

   signal(SIGALRM,ding);
   siginterrupt(SIGALRM,1);  /* don't restart */
   for(;;){
      printf("environ variable name: ");
      alarm(SECS);
      status = scanf("%s",var);
      if (status  < 0 ) {
         if (errno == EINTR && timedout){
            printf("\ntimed out\n");
            exit(0);
         } else {
            perror("scanf failed");
            exit(1);
         }
      }
      alarm(0);  /* turn off timer */
      val = getenv(var);
      if (val)printf("%s = %s\n",var,val);
   }
}
```

Critical region

```
oldmask = sigblock(sigmask(SIGQUIT)
   | sigmask(SIGINT));
/* ...  critical region .... */
sigsetmask(oldmask);
```

see *sigaction()/sigprocmask()*

next time

concurrent servers: fork()