Today

- xti
- streams
- assignments 7-10

XTI

X/Open Transport Interface

- different network API
- conflict between Sys V and BSD
- assumed OSI would replace TCP/IP
- derived from Sys V TLI ('88)
- limited use

TLI

Transport Layer Interface

- Sys V.3, '86 (streams, no TCP/IP)
- interface to transport layer of OSI
- modeled after ISO Transport Service Definition
- not a transport provider, just an interface
- Sys V.4 ('90) provided TCP/IP transport
- supports multiple transports
- transport-independent applications ? NOT
- alternative to BSD sockets
- Sun OS 4.x and Solaris also provide TLI
- OSF provides it via XTI

XTI

X/Open Transport Interface

- superset of TLI
- functions begin with t_
- applications portable across systems, maybe
- mandatory and optional functions
- initally ISO connecition-oriented and connectionless, but now UDP and TCP too.
- can talk to any TCP/UDP client/servers
- Posix.1g standard started with XTI
- */etc/netconfig* defines XTI name and address mapping for each proto

XTI calls

- function calls, NOT system calls
- implemented as part of streams
- both synch and asynch versions, out-of-band data, abortive disconnect, listen queue
- can actually reject a connection
- usually return 0 if OK, -1 if not
- **t_error** for error reporting (like perror)
- some structure filling too
- addressing stuff different for each transport
- uses a generic structure for arguments

```
   struct netbuf {
 unsigned int maxlen;   /* max size of buf */
 unsigned int len;   /* actual size of buf */
 char *buf;       /* protocol dependent data */
   }
```


XTI calls

```
int t_open(char *path, int oflag, struct t_info *out)
  like socket(), returns a "fd"
   path   "/dev/tcp" or udp icmp
   oflag, e.g. O_RDWR
   out returns info on tranportation provider
      -1 no limit,   -2 not supported, max size

int t_bind(fd, struct t_bind *in, struct t_bind *out)
   like bind, specify address for local endpoint
     structure provides listen queue len too
   out, returns info on address actually assigned

int t_connect(fd, struct t_call *server, struct t_call *out)
   connect to server address,
   out contains info on peer, maybe connection data

int t_listen(fd, struct t_call *out)
    WAITs for connect
    out returns info on connection, peer addr.
    does NOT accept connection

int t_accept(fd, connfd, struct t_call *out)
   like accept, fd is the listen fd, connfd (same as fd,
    or previously created) will be used for subsequent comm.
   (Note, can reject connection with t_snndis())
```


XTI calls

To use read/write, you must POP,PUSH tirrdwr stream module.

```
t_snd(fd, char *buff, nbytes, int flags)
t_rcv(fd, char *buff, nbytes, int *flags)

 flags for out-of-band data, or T_MORE for record building
 send/recv data on connection-oriented transport
 both return number of bytes read or written

t_sndudata(fd, struct t_unitdata *u)
t_rcvudata(fd, struct t_unitdata *u, int *flags)

 send/recv data on connectionless transport
 note functions return 0 if OK, -1 if not
 flags can indicate not all data recv'd

    struct t_unitdata {
        struct netbuf addr;   /* protocol-specific addr */
        struct netbuf opt ;   /* protocol-specific opt  */
        struct netbuf udata;  /* user data */
    }

t_alloc(fd, int structtype, int fields)
   allocate data structures, and select which fields
     of structure to allocate
   data structures in user space (not part of kernel)
   by-product of streams implementation
   may need t_sync() for exec() or dup()
```


XTI TCP client

solaris: gcc -o tcpcli tcpcli.c -lnsl

```
...
   char                    *t_alloc();     /* TLI function */
   struct t_call           *callptr;
   struct sockaddr_in      serv_addr;
   if ( (tfd = t_open("/dev/tcp", O_RDWR, 0)) < 0)
           err_sys("client: can't t_open %s", "tcp");

   if (t_bind(tfd, (struct t_bind *) 0, (struct t_bind *) 0) < 0)
           err_sys("client: t_bind error");

   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 ( (callptr = (struct t_call *) t_alloc(tfd, T_CALL, T_ADDR)) =
           err_sys("client: t_alloc error");
   callptr->addr.maxlen = sizeof(serv_addr);
   callptr->addr.len    = sizeof(serv_addr);
   callptr->addr.buf    = (char *) &serv_addr;
   callptr->opt.len     = 0;               /* no options */
   callptr->udata.len   = 0;               /* no user data with conn
   if (t_connect(tfd, callptr, (struct t_call *) 0) < 0)
           err_sys("client: can't t_connect to server");
```

doit()

```
...
   while (fgets(sendline, MAXLINE, fp) != NULL) {
           n = strlen(sendline);
           if (t_snd(tfd, sendline, n, 0) != n)
                   err_sys("client: t_snd error");

           n = t_rcv(tfd, recvline, MAXLINE, &flags);
           if (n < 0)
                   err_sys("client: t_rcv error");
           recvline[n] = 0;        /* null terminate */
           fputs(recvline, stdout);
   }
```


XTI TCP server

```
...
   if ( (tfd = t_open("/dev/tcp", O_RDWR, (struct t_info *) 0)) < 0)
           err_sys("server: can't t_open %s", "tcp");
   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(7654);

   req.addr.maxlen = sizeof(serv_addr);
   req.addr.len    = sizeof(serv_addr);
   req.addr.buf    = (char *) &serv_addr;
   req.qlen        = 5;

   if (t_bind(tfd, &req, (struct t_bind *) 0) < 0)
           err_sys("server: can't t_bind local address");

   /*
    * Allocate a t_call structure for t_listen() and t_accept().
    */

   if ( (callptr = (struct t_call *) t_alloc(tfd, T_CALL, T_ADDR)) =
           err_sys("server: t_alloc error for T_CALL");
```



```
   for ( ; ; ) {
           /*
            * Wait for a connection from a client process.
            * This is an example of a concurrent server.
            */

           if (t_listen(tfd, callptr) < 0)
                   err_sys("server: t_listen error");

           newtfd = t_open("/dev/tcp", O_RDWR, (struct t_info *) 0);
           t_accept(tfd,newtfd,callptr);
           ioctl(newtfd, I_PUSH, "tirdwr"); /* use read/write */

           if ( (childpid = fork()) < 0)
                   err_sys("server: fork error");

           else if (childpid == 0) {       /* child process */
                   t_close(tfd);           /* close original endpoint */
                   str_echo(newtfd);       /* process the request */
                   exit(0);
           }
```

see dunigan/cs494-unp/xtcpclic xtcpserv.c

SYS v streams

- introduced in SVR3, '86, solaris
- not POSIX or BSD, yes Unix 98
- terminal I/O, sockets, and XTI implemented on top of streams
- full-duplex interface between process and a driver
- modularity and flexibility for the I/O subsystem
- allow user to develop "driver interfaces"
- allow devices to share modules
- intermediate processing modules may be added (PUSH'd) or POP'd between stream head and driver
- modules provide filtering on messages moving through the stream
- the head, driver, and modules each contain read/write queues with priority within q's
- three service interfaces for networking modules: TPI, NPI, DLPI
- ioctl() used to PUSH/POP modules
- getmsg/putmsg recv/send data and control info

tracing

- *truss* is used to trace calls on Solaris
- can observe stream calls
- socket() not a "system" call
- connect() is putmsg/getmsg

```
truss techocli 128.169.94.59

...
open("/etc/netconfig", O_RDONLY)              = 3
read(3, " #\n #   T h e   " N e t"., 8192)    = 1064
...
close(3)                                      = 0
open("/dev/tcp", O_RDWR)                      = 3
ioctl(3, I_PUSH, "sockmod")                   = 0
ioctl(3, I_STR, 0xDFFFF128)                   = 0
...
putmsg(3, 0xDFFFF024, 0x00000000, 0)          = 0
getmsg(3, 0xDFFFF024, 0x00000000, 0xDFFFEF1C) = 0
...
write(3, "DF ^\0\0\001 p\0\0\0\0\0"., 1024)   = 1024
read(3, "DF ^\0\0\001 p\0\0\0\0\0"., 1024)    = 1024
write(3, "DF ^\0\0\001 p\0\0\0\0\0"., 1024)   = 1024
read(3, "DF ^\0\0\001 p\0\0\0\0\0"., 1024)    = 1024
```


UDP echo client

sendto/recvfrom

```
truss uechocli menkar
...
open("/dev/udp", O_RDWR)                          = 4
ioctl(4, I_PUSH, "sockmod")                       = 0
ioctl(4, I_STR, 0xDFFFB9C0)                       = 0
...
alarm(3)                                          = 0
putmsg(4, 0xDFFFBA3C, 0xDFFFB9A0, 0)              = 0
getmsg(4, 0xDFFFBA30, 0xDFFFBA3C, 0xDFFFBA1C)     = 0
alarm(0)                                          = 3
alarm(3)                                          = 0
putmsg(4, 0xDFFFBA3C, 0xDFFFB9A0, 0)              = 0
getmsg(4, 0xDFFFBA30, 0xDFFFBA3C, 0xDFFFBA1C)     = 0
alarm(0)                                          = 3
```


next time

review