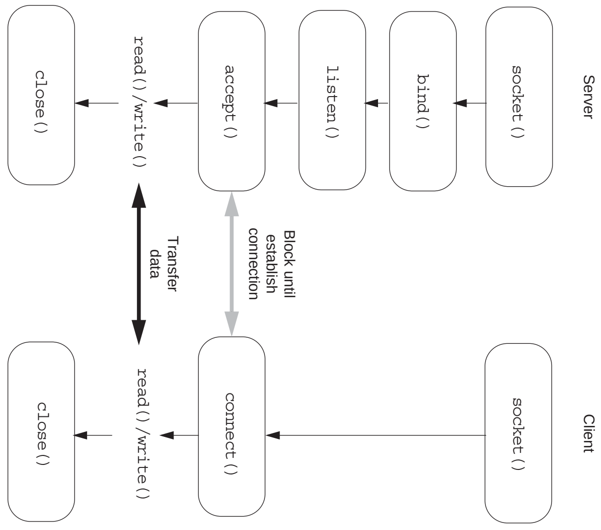
# Module 12

## Sockets

## Overview

- Objectives
- Relevance

## Client-Server Model



## Socket Types

`SOCK_STREAM` – TCP

- Connection-oriented
- No message boundaries
- Reliable
- Sequenced
- Easier to use with reliability, but more expensive

`SOCK_DGRAM` – UDP

- Connectionless
- Message boundaries
- Not reliable, not sequenced
- Duplicates possible
- More efficient

## Combining Socket Domains and Types

- `AF_UNIX`, `SOCK_STREAM`
- `AF_UNIX`, `SOCK_DGRAM`
- `AF_INET`, `SOCK_STREAM`
- `AF_INET`, `SOCK_DGRAM`

## Combining Socket Domains and Types

### Owner's Activities

| Human Speak | Computer Speak | System Call |
|---|---|---|
| Buy a phone | Establish an end point | `socket(3SOCKET)` |
| Get a phone number | Establish a rendezvous | `bind(3SOCKET)` |
| Activate the phone line. | Set queue length and enable service | `listen(3SOCKET)` |
| Wait for a new client and redirect to Cesare's phone. | Acknowledge | `accept(3SOCKET)` and `fork(2)` |
| Cesare takes order | Full duplex conversation | `read(2)`, `write(2)`, `recv(3SOCKET)`, and `send(3SOCKET)` |

## Combining Socket Domains and Types

### Client's Activities

| Human Speak | Computer Speak | System Call |
|---|---|---|
| Walk to any phone. | Establish an end point | `socket()` |
| Look for "Pizza" in phone book | Use directory services | `gethostbyname(3NSL)` |
| Dial the number | Request connection | `connect(3SOCKET)` |
| Place order | Full duplex conversation | `read()`, `write()`, `recv()`, and `send()` |
| Hang up | Send EOF | `close(2)` |

## Combining Socket Domains and Types

### `AF_UNIX, SOCK_STREAM` Call Sequences

| Server | Client |
|---|---|
| `sd = socket` | `sd = socket` |
| `bind` | |
| `listen` | |
| `nsd = accept` | `connect` |
| `write, read, send, recv` | `write, read, send, recv` |
| `close(nsd)` | `close(sd)` |
| `close(sd)` | |

## Combining Socket Domains and Types

### AF_UNIX, SOCK_DGRAM Call Sequences

| Server | Client |
|---|---|
| `sd = socket` | `sd = socket` |
| `bind` | `bind` |
| `recvfrom, sendto` | `sendto, recvfrom` |
| `close(sd)` | `close(sd)` |

## Combining Socket Domains and Types

### AF_INET, SOCK_STREAM Call Sequences

| Server | Client |
|---|---|
| `sd = socket` | `sd = socket` |
| `bind` | |
| `listen` | `gethostbyname` |
| `nsd = accept` | `connect` |
| `write, read, send, recv` | `write, read, send, recv` |
| `close(nsd)` | `close(sd)` |
| `close(sd)` | |

## Combining Socket Domains and Types

### AF_INET, SOCK_DGRAM Call Sequences

| Server | Client |
|---|---|
| `sd = socket` | `sd = socket` |
| `bind` | `bind` |
| | `gethostbyname` |
| `recvfrom, sendto` | `sendto, recvfrom` |
| `close(sd)` | `close(sd)` |

## Creating and Destroying Sockets

- `socket()`– Creates a socket
- `shutdown()`– Destroys a socket

## Binding an Address to a Socket

- `bind()`– Assigns a local address (or name) to the socket descriptor
- `listen()`– Activates a socket

## Binding an Address to a Socket

### UNIX Example

```
int  sd;
struct sockaddr_un name;

sd = socket(AF_UNIX, SOCK_STREAM, 0);
name.sun_family = AF_UNIX;
strcpy(name.sun_path, "/tmp/socket1");
unlink( name.sun_path ); /* Otherwise bind could fail */

if( bind(sd, (struct sockaddr*)&name, sizeof(name) ) == -1 )
{

  perror("bind");
  exit(1);
}
```

## Binding an Address to a Socket

### Internet Example

```
struct sockaddr_in sin;
int sd;

sd = socket(AF_INET, SOCK_STREAM, 0);
memset((char *)&sin, '\0', sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_port = htons(7000); /* pick */
sin.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(sd, (struct sockaddr *)&sin, sizeof(sin))
        == -1) {
   perror("bind");
  exit(1);
}

if (listen(sd, 5) == -1) {
   perror("listen?");
   exit(1);
}
```

## Accepting a Connection

```
struct sockaddr_un client;
int len = sizeof(client), comm;
.....

loop:
while ((comm = accept(sd, &client, &len)) != -1) {
  if (fork() == 0) {
    /* child to handle session */
    close(sd); /* do not need rendezvous */
    do_service(comm); /* talk to client using
        comm */
    close(comm);
    exit(0);
  }
  close(comm); /* do not need communication */
}
```

```
/* If a signal arrives during accept(), it fails
with -1. So, if catching signals, use the following
2 lines */
if (errno == EINTR) {
  goto loop;
}
```

## Connecting to the Server

### Client Side Example

```
int sd;
struct sockaddr_un  name;

sd = socket(AF_UNIX, SOCK_STREAM, 0);
name.sun_family = AF_UNIX;
strcpy( name.sun_path, "/tmp/socket1");

if (connect(sd, (struct sockaddr *)&name,
        sizeof(name)) == -1) {
  perror("connect");
  exit(1);
}
```

## Connecting to the Server

### Internet Socket Example

```
int sd;
struct sockaddr_in sin;
struct hostent *he;

sd = socket(AF_INET, SOCK_STREAM, 0);

memset((char *)&sin, '\0', sizeof(sin));
sin.sin_family = AF_INET;

/* match server */
he = gethostbyname("Server");
sin.sin_port = htons(7000);
memcpy((char *)&sin.sin_addr, he->h_addr_list[0],
he->h_length);

```

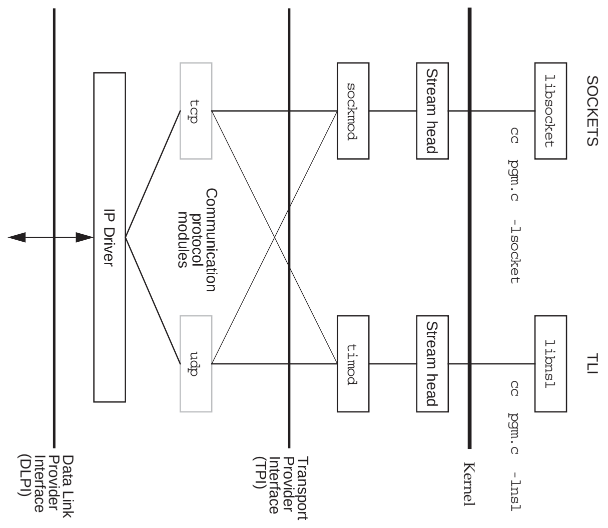
```
if (connect(sd, (struct sockaddr *)&sin,
      sizeof(sin)) == -1) {
  perror("connect");
  exit(1);
}
```

## Receiving Data

```
char message[80];
int length = sizeof(message);
int rval, i;

if ((rval = recv(sd, message, length, 0)) == -1) {
    perror("recv");
  exit(1);
}
for (i = 0; i < rval; i++) {
  putchar(message[i]);
}
```

## Network Sockets and TLI



## Network Sockets and TLI

| | Sockets | TLI |
|---|---|---|
| Performance | Comparable – both use STREAMS | Comparable - both use STREAMS |
| POSIX | Accepted for adoption | Accepted for adoption |
| Transport independence | No | Yes, but must purchase unbundled software if you want to support other than TCP and UDP |

## Establishing a Connect to a Remote Host

```
/* gethost.c */

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

main(int argc, char *argv[]){

  struct hostent *he;
  char    **tmp;

  if (argc != 2) {
    fprintf(stderr, "Usage: %s hostname\n", argv[0]);
    exit(1);
  }
  he = gethostbyname(argv[1]);

  if (he == NULL) {
    fprintf(stderr, "No such host.\n");
    exit(1);
  }
  printf("Official name of %s is %s.\n",
         argv[1], he->h_name);

  for (tmp=he->h_aliases; *tmp != NULL; tmp++) {
    printf("\tAlias: %s\n", *tmp);
  }

  tmp = he->h_addr_list;
  printf("IP address: %d.%d.%d.%d\n",
            (unsigned char)(*tmp)[0],
            (unsigned char)(*tmp)[1],
            (unsigned char)(*tmp)[2],
            (unsigned char)(*tmp)[3]);
  return 0;
}
```

## Determining File Descriptor Activity

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <poll.h>
#include <errno.h>

/* Number of file descriptors to poll. */
#define NUM_FDS_TO_POLL 2

main() {

  int udpsd1, udpsd2;
  struct sockaddr_in sin1;
  struct sockaddr_in sin2;
  struct sockaddr_in returnAddr;
  int returnAddrSize;

```

```
  /* This array of struct pollfd defines fully
  what poll() is to wait on, and provides a place
  for poll() to respond. */
  struct pollfd fds[NUM_FDS_TO_POLL];

  /* Set up two UDP sockets to wait on. */

  memset((char *)&sin1, ‘\0’, sizeof(sin1));
  sin1.sin_family = AF_INET;
  sin1.sin_port = htons(7000);
  sin1.sin_addr.s_addr = htonl(INADDR_ANY);

  /* create one udp sd */
  udpsd1 = socket(AF_INET, SOCK_DGRAM, 0);
  if (udpsd1 == -1) {
    perror (“udp socket”);
    exit (errno);
  }

```

```
  if (bind(udpsd1, (struct sockaddr *)&sin1,
    sizeof(sin1)) == -1) {
    perror (“tcp bind”);
    exit (errno);
  }

  memset((char *)&sin2, ‘\0’, sizeof(sin2));
  sin2.sin_family = AF_INET;
  sin2.sin_port = htons(7001);
  sin2.sin_addr.s_addr = htonl(INADDR_ANY);

  /* create one udp sd */
  udpsd2 = socket(AF_INET, SOCK_DGRAM, 0);
  if (udpsd2 == -1) {
    perror (“udp socket”);
    exit (errno);
  }

```

```
if (bind(udpsd2, (struct sockaddr *)&sin2,
  sizeof(sin2)) == -1) {
  perror (“udp bind”);
  exit (errno);
}

/*
Initialize each element of the fds array with:
  - fd: the file descriptor of interest.
  - events: what type of event to wait on.

Poll returns with the revents field set with the
events which are ready to do.  E.G. if it
returns POLLIN, a read can be done without
blocking.
*/

fds[0].fd = udpsd1;
fds[0].events = POLLIN;
fds[1].fd = udpsd2;
fds[1].events = POLLIN;

```

```
/*
Timeout value (poll’s third argument) can be:
 0: Immediate return whether or not fd is ready.
  -1 (INFTIM): Blocks until a fd is ready.
  <other value>: time in mS to wait for a fd
  before returning.
*/

for (;;) {
  int numfds;
  if ((numfds = poll(fds, NUM_FDS_TO_POLL, -1)) < 0) {
    perror (“poll”);
    exit (errno);
  } else {
    printf (“Poll returned %d fds to read.\n”,
      numfds);
  }

```

```
  /* if udp sd ready, use it */
  if (fds[0].revents == POLLIN) {
    char buffer[80];
    if (recvfrom(udpsd1, buffer, 80, 0,
      (struct sockaddr *)&returnAddr,
      &returnAddrSize) >= 0) {
    puts(buffer);
    } else {
    perror (“recvfrom”);
    }
  }

```

```
    /* if udp sd ready, use it */
    if (fds[1].revents == POLLIN) {
      char buffer[80];
      if (recvfrom(udpsd2, buffer, 80, 0,
        (struct sockaddr *)&returnAddr,
        &returnAddrSize) >= 0) {
      puts(buffer);
      } else {
      perror (“recvfrom”);
      }
    }
  }
}

```

## Asynchronous I/O

```
int char_present;

void handler() {
  char_present++;
}

int fd, flags;

/* Install handler for SIGPOLL, see Signals module*/
    install_disp(SIGPOLL, handler);

/* Register fd to have SIGPOLL sent when IO is
 * possible. Usually fd is a pipe or socket. */
  ioctl(fd, I_SETSIG, S_RDNORM);

  while(1) {

```

```
  /* Block SIGPOLL to prevent char_present being changed */
  if (char_present) {

    /* Can use poll(2) to determine which
    descriptor sent the SIGPOLL, if more than one. */
    do_io();
    char_present = 0;
  }

  /* Unblock SIGPOLL */
  .
  .  /* Main program work is in this loop */
  .
}
```

## `AF_UNIX,SOCK_STREAM` Example

### Server Side

```
/* srvr1.c -  AF_UNIX, SOCK_STREAM
 * Server reads a value from client and
 * sends back twice that value */

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

  main() {

  int sd, comm;
  struct sockaddr_un myname, client;
  int namesize = sizeof(struct sockaddr_un);
  int clientsize = sizeof(struct sockaddr_un);
  double dub;

```

```
  /* Create server's rendezvous socket sd */
  if( (sd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
    perror("srvr1.c:main:socket");
    exit(1);
  }

  /* Fill in server's address and bind it to sd */
  myname.sun_family = AF_UNIX;
  strcpy(myname.sun_path, "/tmp/socket1");
  unlink( myname.sun_path );

  if( bind(sd, (struct sockaddr*)&myname, namesize)
            == -1 ) {
    perror("srvr1.c:main:bind");
    exit(1);
  }

  /* Prepare to receive multiple connect requests */
  if( listen(sd, 128) == -1 ) {
    perror("srvr1.c:main:listen");
    exit(1);
  }

  /* Infinite loop to accept client requests */
  while(1) {
    comm = accept(sd, (struct sockaddr*)&client,
        &clientsize);
    if( comm  == -1) {
      perror("srvr1.c:main:accept");
      exit(1);
    }
    read( comm, &dub, sizeof(dub));
    dub += dub;
    write( comm, &dub, sizeof(dub));
    close(comm);
  }

  /* Install a signal handler for cleanup including:
   * close(sd);
   * unlink( myname.sun_path );  */
}
```

## Client Side

```
/* clnt1.c - AF_UNIX, SOCK_STREAM */

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

main() {

  int namesize = sizeof(struct sockaddr_un);
  int sd;
  struct sockaddr_un srvr;
  double dub=2.3, dub2;

  /* Create client's socket sd */
  if( (sd = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
    perror("clnt1.c:main:socket");
    exit(1);
  }

```

```
  /* Fill in server's address and connect to server */
  srvr.sun_family = AF_UNIX;
  strcpy(srvr.sun_path, "/tmp/socket1");

    if( connect(sd, (struct sockaddr*)&srvr, namesize)
      == -1 ) {
    perror("clnt1.c:main:connect");
    exit(1);
  }

  /* Communicate with server */
  write(sd, &dub,  sizeof(dub));
  read (sd, &dub2, sizeof(dub2));
  close(sd);
  printf("dub: %.2f, dub2: %.2f\n", dub, dub2);
}
```

## `AF_UNIX,SOCK_DGM` Example

### Server Side

```
/* srvr2.c - AF_UNIX, SOCK_DGRAM
 * Server reads a string from client and
 * sends back a greeting appended to the string */

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

#define BUFSZ  256

main() {

  int sd, comm;
  struct sockaddr_un myname, client;
  int namesize = sizeof(struct sockaddr_un);
  char buf[BUFSZ];

  /* Create server's socket sd */
  if( (sd = socket(AF_UNIX, SOCK_DGRAM, 0)) == -1) {
    perror("srvr2.c:main:socket");
    exit(1);
  }

  /* Fill in server's address and bind it to sd */
  myname.sun_family = AF_UNIX;
  strcpy(myname.sun_path, "/tmp/socket2srvr");
  unlink( myname.sun_path );

  if( bind(sd, (struct sockaddr*)&myname, namesize)
           == -1 ) {
    perror("srvr2.c:main:bind");
    exit(1);
  }

```

```
  /* Infinite loop to accept client requests */
  while(1){
    if( recvfrom(sd, (char*)&buf, BUFSZ, 0,
        (struct sockaddr*)&client, &namesize)  == -1) {
      perror("srvr2.c:main:recvfrom");
      exit(1);
    }
    strcat(buf, ", Hello from the server" );

    if( sendto(sd, buf, BUFSZ, 0,
        (struct sockaddr*)&client, namesize)  == -1) {
      perror("srvr2.c:main:sendto");
      exit(1);
    }
  }

  /* Install a signal handler for cleanup including:
   * close(sd);
   * unlink( myname.sun_path );  */
}
```

## Client Side

```
/* clnt2.c - AF_UNIX, SOCK_DGRAM */

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

#define BUFSZ 256

main() {

  int namesize = sizeof(struct sockaddr_un);
  int sd;
  struct sockaddr_un clnt, srvr;
  char buf[BUFSZ];

  /* Create client's socket sd */
  if( (sd = socket(AF_UNIX, SOCK_DGRAM, 0)) == -1) {
    perror("clnt2.c:main:socket");
    exit(1);
  }
```

```

  /* Fill in client's address and bind it to sd */
  clnt.sun_family = AF_UNIX;
  strcpy(clnt.sun_path, "/tmp/socket2clnt");
  unlink( clnt.sun_path );

  if( bind(sd, (struct sockaddr*)&clnt, namesize)
     == -1 ) {
    perror("clnt2.c:main:bind");
    exit(1); }

  /* Fill in server's address for sendto server */
  srvr.sun_family = AF_UNIX;
  strcpy(srvr.sun_path, "/tmp/socket2srvr");

  /* Communicate with server */
  if(sendto(sd,"Deac",5,0,(struct sockaddr*)&srvr,
    namesize)==-1) {
    perror("clnt2.c:main:sendto");
    exit(1);
  }

  if( recvfrom(sd,buf,BUFSZ,0,(struct sockaddr*)&srvr,
        &namesize) ==-1) {
    perror("clnt2.c:main:recvfrom");
    exit(1);
  }
  printf("CLIENT REC'D: %s\n", buf);
  close(sd);
  unlink( clnt.sun_path );
}
```

## AF_INET, SOCK_STREAM Example

### Server Side

```
/* srvr3.c - Server - full example
 * AF_INET, SOCK_STREAM Time Server
 * Server sends its current time */

  #include <stdio.h>
  #include <sys/types.h>
  #include <sys/socket.h>
  #include <sys/utsname.h>
  #include <netdb.h>
  #include <netinet/in.h>
  #include <errno.h>
  #include <time.h>

  #define PORTNUM 5998

```

```
main() {

struct utsname name;
struct sockaddr_in socketname, client;
int sd, ns, clientlen = sizeof(client);
struct hostent *host;
time_t today;

/* determine server system name and internet address */
if (uname(&name) == -1) {
  perror("uname");
  exit(1);
}

if ((host = gethostbyname(name.nodename)) == NULL) {
  perror("gethostbyname");
  exit(1);
}

```

```
/* fill in socket address structure */
memset((char *) &socketname, '\0', sizeof(socketname));
socketname.sin_family = AF_INET;
socketname.sin_port = PORTNUM;
memcpy( (char *) &socketname.sin_addr,
      host->h_addr,host->h_length);

/* open socket */
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
  perror("socket");
  exit(1);
}

/* bind socket to a name */
if (bind(sd, (struct sockaddr *) & socketname,
         sizeof(socketname))) {
  perror("bind");
  exit(1);
}

```

```
    /* prepare to receive multiple connect requests */
    if (listen(sd, 128)) {
      perror("listen");
      exit(1);
    }

    while (1) {
      if ((ns = accept(sd, (struct sockaddr *)&client,
                       &clientlen)) == -1) {
        perror("accept");;
        exit(1);
      }

      /* get current time and date */
      time(&today);

```

```
      /* send time through socket */
      if(write(ns, &today, sizeof(today) ) == -1) {
        perror("write");
        exit(1);
      }
      close(ns);
    }
}
```

## Client Side

```
/* clnt3.c - Client full example
 * AF_INET, SOCK_STREAM Client of Time Server */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/utsname.h>
#include <netdb.h>
#include <netinet/in.h>
#include <errno.h>
#include <time.h>

#define PORTNUM 5998

main() {

  int sd;
  struct sockaddr_in server;
  struct hostent *host;
  time_t srvrtime;
```

```

  struct utsname name;
  /* create the socket for talking to server*/
  if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }

  /* get server internet address and put into addr
   * structure fill in the socket address structure
   * and connect to server */
  memset((char *) &server, '\0', sizeof(server));
  server.sin_family = AF_INET;
  server.sin_port = PORTNUM;

  /* Server is local system. Get its name. */
  if (uname(&name) == -1) {
    perror("uname");
    exit(1);
  }
  if ((host = gethostbyname("name.nodename")) == NULL) {
    perror("gethostbyname");
    exit(1);
  }
  memcpy((char *)&server.sin_addr, host->h_addr,
          host->h_length);

  /* connect to server */
  if( connect(sd, (struct sockaddr *)&server,
            sizeof(server))) {
    perror("connect");
    exit(1);
  }

  /* read the time and date passed from the server */
  if(read(sd, &srvrtime, sizeof(srvrtime) ) == -1) {
    perror("recv");
    exit(1);
  }
  close(sd);

  printf("Server's time is: %s", ctime(&srvrtime));
}
```

## Exercise: Sockets

- Objectives
- Tasks
- Discussion
- Solutions