Socket Interface

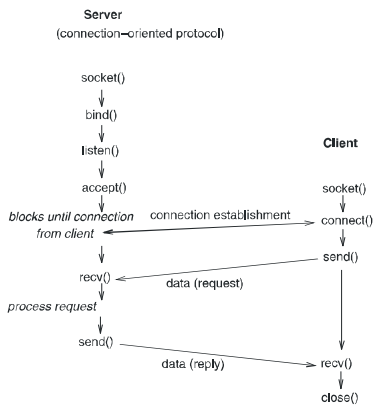
- Interface between application programs and TCP/IP software (introduced in Berkeley UNIX Operating System)
- Centers around *socket abstraction*
- Follows open-read-write-close paradigm

- socket (endpoint) = <IP address, port number>

Connection-oriented (TCP) Based Application

- Server Program
  - Create a socket
  - Bind it to a well-known port on local machine
  - Wait for clients

- Client Program
  - Create a socket
  - Connect it to a server on a remote machine
  - Use it to send/receive data to/from remote machine
  - When done, close socket



Socket Operations

- Creating a socket

  `int socket(int domain, int type, int protocol)`
  - domain=AF_INET (for TCP/IP protocols)
  - type=SOCK_STREAM (for TCP-based application)
- Passive open on server

  `int bind(int socket, struct sockaddr *address, int addr_len)`

  `int listen(int socket, int backlog)`

  `int accept(int socket, struct sockaddr *address, int addr_len)`

Socket Operations (cont'd)

- Active open on client

  `int connect(int socket, struct sockaddr *address, int addr_len)`
- Sending and receiving messages

  `int send(int socket, char *message, int msg_len, int flags)`

  `int recv(int socket, char *buffer, int buf_len, int flags)`

Concurrent Server Implementation

Master server does the following:

- Open port:
  - master opens well-known port
- Wait for client:
  - master waits for a new client request
- Start Slave:
  - master starts a slave to handle request (e.g., in UNIX/Linux, it ***forks*** a copy of the server process)
  - master closes ``connected'' socket, and slave closes ``listening'' socket
  - slave dies when done
- Continue:
  - master returns to the ***wait*** step

Server Code: establish socket

```
/* code to establish a socket */
int establish(unsigned short portnum) {
      char   myname[MAXHOSTNAME+1];
      int    s;
      struct sockaddr_in sa;
      struct hostent *hp;

      memset(&sa, 0, sizeof(struct sockaddr)); /* clear our address */
      gethostname(myname, MAXHOSTNAME);    /* who are we? */
      hp= gethostbyname(myname);                    /* get our address info */
      if  (hp == NULL)                        /* we don't exist !? */
            return(-1);
      sa.sin_family= hp->h_addrtype;    /* this is our host address */
      sa.sin_addr = htonl(INADDR_ANY);
      sa.sin_port= htons(portnum);        /* this is our port number */
      if ((s= socket(AF_INET, SOCK_STREAM, 0)) < 0)  /* create socket */
            return(-1);
      if (bind(s,(struct sockaddr *)&sa,sizeof(sa))) < 0) {
            close(s);
            return(-1);                /* bind address to socket */
      }
      listen(s, 3);               /* max # of queued connects */
      return(s); }
```


Server Code: wait for clients

```
/* wait for a connection to occur on a socket created with establish() */
int get_connection(int s) {
      int t;           /* socket of connection */

      if ((t = accept(s,NULL,NULL)) < 0)   /* accept connection if there is one */
        return(-1);
      return(t); }
```


Server Code: main program

```
#include <errno.h>      /* obligatory includes */
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <netdb.h>

#define PORTNUM 5000 /* random port number, we need something */
void do_something(int);

main() {
      int s, t;

      if ((s = establish(PORTNUM)) < 0) {  /* plug in the phone */
            perror("establish");
            exit(1);
      }
```

Server Code: main (cont' d)

```
    for (;;) {              /* loop for phone calls */
        if  ((t= get_connection(s)) < 0) { /* get a connection */
            perror("accept");      /* bad */
            exit(1);
        }
        do_something(t);
        close(t);                  /* another connection */
        continue;
    }
} /* end of main */

/* this is the function that plays with the socket.  It will
   be called after getting a connection. */
void do_something(int t) {
    /* do your thing with the socket here */
}
```

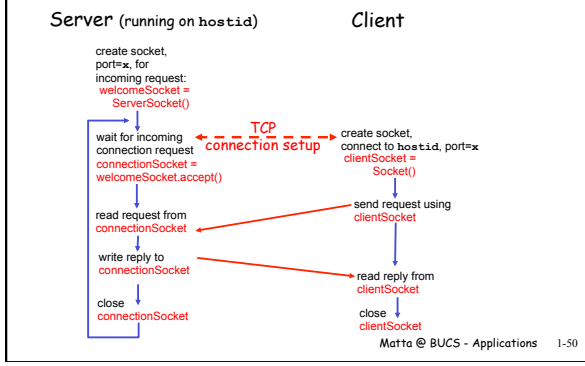

Concurrent Server

```
/* how a concurrent server looks like */

for (;;) {                    /* loop for phone calls */
    if ((t= get_connection(s)) < 0) {       /* get a connection */
        perror("accept");           /* bad */
        exit(1);
    }
    switch( fork() ) {              /* try to handle connection */
    case -1 :                    /* bad news.  scream and die */
        perror("fork");
        close(s);
        close(t);
        exit(1);
    case 0 :                     /* we're the child, do something */
        close(s);
        do_something(t);
        close(t);
        exit(0);
    default :                    /* we're the parent so look for */
        close(t);               /* another connection */
        continue;
    }
}
```


Client Code

```
int call_socket(char *hostname, unsigned short portnum) {
    struct sockaddr_in sa;
    struct hostent    *hp;
    int a, s;

    if  ((hp= gethostbyname(hostname)) == NULL) { /* do we know the host's address?*/
        return(-1);   /* no */
    }
    memset(&sa,0,sizeof(sa));
    memcpy((char *)&sa.sin_addr,hp->h_addr,hp->h_length);  /* set address */
    sa.sin_family= hp->h_addrtype;
    sa.sin_port= htons(portnum);
    if ((s= socket(hp->h_addrtype,SOCK_STREAM,0)) < 0)     /* get socket */
        return(-1);
    if (connect(s,(struct sockaddr *)&sa,sizeof(sa)) < 0) { /* connect */
        close(s);
        return(-1);
    }
    return(s);
}
```

Java client/server socket interaction: TCP



Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

   public static void main(String argv[]) throws Exception
   {
      String sentence;
      String modifiedSentence;

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      Socket clientSocket = new Socket("hostname", 6789);

      DataOutputStream outToServer =
        new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

```
      BufferedReader inFromServer =
        new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));

      sentence = inFromUser.readLine();

      outToServer.writeBytes(sentence + '\n');

      modifiedSentence = inFromServer.readLine();

      System.out.println("FROM SERVER: " + modifiedSentence);

      clientSocket.close();

   }
}
```

Create input stream attached to socket

Send line to server

Read line from server

Example: Java server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
  {
    String clientSentence;
    String capitalizedSentence;

    ServerSocket welcomeSocket = new ServerSocket(6789);

    while(true) {

      Socket connectionSocket = welcomeSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Example: Java server (TCP), cont

```
      DataOutputStream  outToClient =
        new DataOutputStream(connectionSocket.getOutputStream());

      clientSentence = inFromClient.readLine();

      capitalizedSentence = clientSentence.toUpperCase() + '\n';

      outToClient.writeBytes(capitalizedSentence);
      connectionSocket.close();
    }
  }
}
```

Create output stream, attached to socket

Read in line from socket

Write out line to socket

End of while loop, loop back and wait for another client connection

Client/server Socket Interaction: UDP