CSC 546: Client/Server Fundamentals

Fall 2000

Sockets

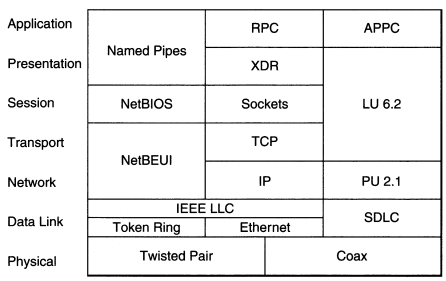
- Internet protocol suite: IP + TCP/UDP + Sockets + RPC/XDR
- file-like interface
- examples:
  - datastream
  - datagram
  - Web client
- Windows sockets

Internet protocol suite

Network layer: Internet Protocol (IP)
- provides generalized packet network interface, handles routing through the Internet

Transport layer: Transmission Control Protocol (TCP)
- provides a virtual circuit for process communication, connection-oriented

Transport layer: User Datagram Protocol (UDP)
- fast, unreliable, connectionless alternative to TCP



Sockets

Sockets provide a session layer protocol

- sits on top of either TCP (datastream) or UDP (datagram)

Sockets were introduced in Berkeley UNIX

- in UNIX, all devices have a similar file-like interface
  applications use commands like `open`, `close`, `read`, `write`
- the socket abstraction is used to provide a similar interface to networks
  when an application creates a socket, obtains a file-like socket descriptor
  communication via `read` and `write`
- sockets can be connection-oriented (via TCP) or connectionless (via UDP)

Sockets for client/server

a socket is like a telephone

- endpoint of a 2-way communication channel

in order for a client & server to communicate

- client & server must each create sockets
- client must know the address of the server (but not vice versa)
- client must establish a connection and identify self to server
- then can communicate via the socket

Using C libraries, can access socket layer directly from applications

Note: the following examples will be in C, UNIX-based

- in C since socket data structures are all low-level, C-like
- in UNIX since simpler (Windows version, WinSock, messier)

Datastream sockets

client must:

1. create a socket using the `socket()` system call
2. connect the socket to the address of the server using `connect()`
3. send and receive data using `read()` and `write()`

server must:

1. create a socket using the `socket()` system call
2. bind the socket to an address (host name + port) using `bind()`
3. listen for connections using `listen()`
4. accept a connection using `accept()`
5. send and receive data using `read()` and `write()`

Client side

client must first create a socket

```
socketDescriptor = socket( addressFormat, type, protocol );
```

`addressFormat` specifies the type of addresses that will be used

- `AF_UNIX` → UNIX domain (for processes that share a common file system)
- `AF_INET` → Internet domain (for processes communicating over the Internet)

`type` specifies the type of socket

- `SOCK_STREAM` → datastream (characters are read in a continuous stream)
- `SOCK_DGRAM` → datagram (connectionless, messages are read in chunks)

`protocol` specifies the communication protocol

- `0` → allows the OS to choose the most appropriate protocol (TCP for datastream, UDP for datagram)

```
int sockfd;
sockfd = socket( AF_INET, SOCK_STREAM, 0);
```

Client side (cont.)

once a socket is created, it must be connected to the server

```
connect( socketDescriptor, destAddress, addressLength );
```

`socketDescriptor` is the int returned by the `socket()` call

`destAddress` specifies the address of the server

- an address is a data structure (as defined in `<netinet/in.h>`)

```
struct sockaddr_in {
    short   sin_family;       /* must be AF_INET     */
    ushort  sin_port;         /* protocol port       */
    struct  in_addr sin_addr; /* IP address          */
    char    sin_zero[8];      /* not used, must be 0 */
```

`addressLength` specifies the length of the address structure (in bytes)

Client side (cont.)

when setting the fields of the server address,

- can use code from `<netdb.h>` to convert a host name to an address

```
struct sockaddr_in serv_addr;                  /* declare address structure */

serv_addr.sin_family = AF_INET;                /* assign family field       */

struct hostent *server;                        /* get host info             */
server = gethostbyname("duck.creighton.edu");
bcopy(server->h_addr,&serv_addr.sin_addr.s_addr,server->h_length);
                                               /* bitwise copy of host addr */

serv_addr.sin_port = htons(52050);             /* port (in host notation)   */
```

note: client and server must agree on a port number

- 16-bit unsigned integer (0 - 65535), with 0-1024 reserved

Client side (cont.)

client sends a message to the server by writing to the socket

```
write( socketDescriptor, data, dataLength );
```

`socketDescriptor` is the int returned by the `connect()` call
`data` is the array of chars to be sent to the server
`dataLength` specifies the length of the data (in bytes)

client receives a message from the server by reading from the socket

```
read( socketDescriptor, buffer, bufferLength );
```

`socketDescriptor` is the int returned by the `connect()` call
`buffer` is an array of chars for storing the data
`bufferLength` specifies the max length of the data that can be read

when done, close the socket using `close()`

Sample client program

1. client reads message from user
2. sends message to server
3. server sends response
4. client displays response locally
5. closes socket

```
printf("Please enter the message: ");   /* prompts user for message    */
bzero(buffer,256);                      /* clears buffer               */
fgets(buffer,255,stdin);                /* reads message into buffer   */
write(sockfd,buffer,strlen(buffer));    /* writes to server via socket */

bzero(buffer,256);                      /* clears buffer               */
read(sockfd,buffer,255);                /* reads response from server  */
printf("%s\n",buffer);                  /* displays locally            */

close(sockfd);                          /* closes the connection       */
```

client.c

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

#define PORT 52050
#define HOST "duck.creighton.edu"

int main()
{
    int sockfd;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[256];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);  /* creates stream socket      */

    serv_addr.sin_family = AF_INET;            /* sets host address structure */
    server = gethostbyname(HOST);
    bcopy(server->h_addr,&serv_addr.sin_addr.s_addr,server->h_length);
    serv_addr.sin_port = htons(PORT);

    connect(sockfd,&serv_addr,sizeof(serv_addr));  /* connects to host       */

    printf("Please enter the message: ");   /* prompts user for message       */
    bzero(buffer,256);                      /* clears buffer                  */
    fgets(buffer,255,stdin);                /* reads message into buffer      */
    write(sockfd,buffer,strlen(buffer));    /* writes to server via socket    */

    bzero(buffer,256);                      /* clears buffer                  */
    read(sockfd,buffer,255);                /* reads response from server     */
    printf("%s\n",buffer);                  /* displays locally               */

    close(sockfd);

    return 0;
}
```

Server side

server must create a socket (as before)

```
int sockfd;
sockfd = socket( AF_INET, SOCK_STREAM, 0);
```

must bind the socket to an address (IP number + port)

```
bind( socketDescriptor, localAddress, addressLength );
```

address is (again) specified using `sockaddr_in` data structure

utilizes `INADDR_ANY` to specify IP of local machine

```
struct sockaddr_in serv_addr;

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(52050);

bind(sockfd, &serv_addr, sizeof(serv_addr));
```

Server side (cont.)

server must then listen for a connection

```
listen( socketDescriptor, queueLength );
```

queueLength specifies the max number of connections allowed

```
listen(sockfd,5);
```

once a connection is attempted, server must accept it

```
newSocketDesc = accept(socketDesc, clientAddr, addrLength);
```

a new server socket is created and connected to the client's socket

```
clilen = sizeof(cli_addr);
newsockfd = accept(sockfd, &cli_addr, &clilen);
```

can then communicate with client using read & write

- close the connection using `close()` when done

Sample server program

Repeatedly:

1. receive message from client
2. display locally
3. send acknowledgement to client
4. close the socket

```
while (1) {
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd, &cli_addr, &clilen);

    bzero(buffer,256);
    read(newsockfd,buffer,255);

    printf("Here is the message: %s\n",buffer);
    write(newsockfd,"I got your message",18);

    close(newsockfd);
  }
```

server.c

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

#define PORT 52050

int main()
{
     int sockfd, newsockfd, clilen;
     char buffer[256];
     struct sockaddr_in serv_addr, cli_addr;

      sockfd = socket(AF_INET, SOCK_STREAM, 0);

     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(PORT);

     bind(sockfd, &serv_addr, sizeof(serv_addr));

     listen(sockfd,5);

     while (1) {
       clilen = sizeof(cli_addr);
       newsockfd = accept(sockfd, &cli_addr, &clilen);

       bzero(buffer,256);
       read(newsockfd,buffer,255);

       printf("Here is the message: %s\n",buffer);
       write(newsockfd,"I got your message",18);

       close(newsockfd);
     }

     return 0;
}
```

Possible extensions

client.c & server.c are based on code from
http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html

- all error checking is removed for simplicity
  a robust implementation would handle bad host names, used ports, lost connections, …
- as is, server cannot handle more than one connection at a time
  a robust implementation would fork off a new process for each accept
  - new process handles the client request
  - server program can then continue to accept other connections

  must then be careful of zombie processes
  - forked processes don't automatically die when done

Datagram sockets

differences from datastreams:

- datagrams are unreliable (if a packet of data gets lost, sender is not told)
  uses UDP transport protocol
- message boundaries are preserved in datagram sockets
  if sender sends 100 bytes, receiver must read all 100 bytes at once
  (using a stream, receiver could read message in pieces)
- communication is done using `receivefrom()` and `sendto()` rather than `read()` and `write()`
- less overhead since a connection does not need to be established then broken, and packets do not need to be acknowledged

as such, datagrams are often used if service to be provided is short

Datagram versions of client.c & server.c

client_udp.c

- `socket()` call uses `SOCK_DGRAM` instead of `SOCK_STREAM`
- there is no `connect()` call
- uses `recvfrom()` and `sendto()` instead of `read()` and `write()`

```
recvfrom(socketDesc, msg, msgLength, options, addr, addrLength);

sendto(socketDesc, msg, msgLength, outofband, addr, addrLength);
```

server_udp.c

- `socket()` call uses `SOCK_DGRAM` instead of `SOCK_STREAM`
- there are no `listen()` or `accept()` calls
- uses `recvfrom()` and `sendto()` instead of `read()` and `write()`

client_udp.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>

#define PORT 52051
#define HOST "duck.creighton.edu"

int main()
{
  int sockfd, length;
  struct sockaddr_in serv_addr, from;
  struct hostent *server;
  char buffer[256];

  sockfd = socket(AF_INET, SOCK_DGRAM, 0);

  serv_addr.sin_family = AF_INET;
  server = gethostbyname(HOST);
  bcopy(server->h_addr,&serv_addr.sin_addr,server->h_length);
  serv_addr.sin_port = htons(PORT);

  printf("Please enter the message: ");
  bzero(buffer,256);
  fgets(buffer,255,stdin);

  length=sizeof(struct sockaddr_in);
  sendto(sockfd,buffer,strlen(buffer),0,&serv_addr,length);

  recvfrom(sockfd,buffer,256,0,&from,&length);
  printf("Got an ack: %s\n", buffer);

  return 0;
}
```

server_udp.c

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

#define PORT 52051

int main()
{
  int sockfd, fromlen;
  struct sockaddr_in server;
  struct sockaddr_in from;
  char buf[1024];

  sockfd = socket(AF_INET, SOCK_DGRAM, 0);

  server.sin_family=AF_INET;
  server.sin_addr.s_addr=INADDR_ANY;
  server.sin_port=htons(PORT);

  bind(sockfd,&server,sizeof(server));

  while (1) {
    bzero(buf, 1024);

    fromlen = sizeof(struct sockaddr_in);
    recvfrom(sockfd,buf,1024,0,&from,&fromlen);

    printf("Received a datagram: %s\n", buf);

    sendto(sockfd,"Got your message\n",17,0,&from,fromlen);
  }

  return 0;
}
```

Implementing a Web browser

can implement a rudimentary Web browser with sockets

- connect socket to port 80 on Web server machine
- send message consisting of HTTP request

```
GET file_name HTTP/1.1
Host: server_name
```

- read and display resulting text (Web page source code)

Note: page may exceed buffer size
need to repeatedly read and display until no more message
(i.e., read returns 0 value)

clientWeb.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>

#define PORT 80
#define HOST "www.creighton.edu"

int main()
{
  int sockfd, status;
  struct sockaddr_in serv_addr;
  struct hostent *server;
  char buffer[256], str[256];

  sockfd = socket(AF_INET, SOCK_STREAM, 0);

  serv_addr.sin_family = AF_INET;
  server = gethostbyname(HOST);
  bcopy(server->h_addr,&serv_addr.sin_addr.s_addr,server->h_length);
  serv_addr.sin_port = htons(PORT);

  connect(sockfd,&serv_addr,sizeof(serv_addr));

  printf("Enter the desired file name: ");
  bzero(buffer,256);
  fgets(buffer,255,stdin);

  bzero(str, 256);
  strcat(str, "GET ");
  strcat(str, buffer);
  strcat(str, "HTTP/1.1\nHost: ");
  strcat(str, HOST);
  write(sockfd,str,strlen(str));

  bzero(buffer,256);
  status = read(sockfd,buffer,255);
  while (status != 0) {
      printf("%s\n",buffer);
      bzero(buffer,256);
      status = read(sockfd,buffer,255);
  }

  return 0;
}
```

Windows sockets

The Winsock library implements UNIX-like sockets for Windows

- must include `<winsock2.h>` instead of UNIX files
- must add the following to the program

```
WSADATA ws;
WSAStartup(0x0101,&ws);
```

- instead of `read` and `write`, use `recv` and `send`
- instead of `bcopy` and `bzero`, use `memcpy` and `memset`
- in Visual C++, must add `Wsock32.lib` to the project :
  `C:\Program Files\Microsoft Visual Studio\Vc98\Lib\Wsock32.lib`

clientWin.c

```
#include <stdio.h>
#include <winsock2.h>

#define PORT 52050
#define HOST "duck.creighton.edu"

int main()
{
    int sockfd;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[256];

    WSADATA ws;
    WSAStartup(0x0101,&ws);

    sockfd = socket(AF_INET, SOCK_STREAM, 0);  /* creates stream socket       */

    serv_addr.sin_family = AF_INET;            /* sets host address structure */
    server = gethostbyname(HOST);
    memcpy(&serv_addr.sin_addr,server->h_addr,server->h_length);
    serv_addr.sin_port = htons(PORT);

    connect(sockfd, (struct sockaddr *)&serv_addr,sizeof(serv_addr));
                                           /* connects to host               */

    printf("Please enter the message: ");  /* prompts user for message       */
    memset(buffer,0,256);                  /* clears buffer                  */
    fgets(buffer,255,stdin);               /* reads message into buffer      */
    send(sockfd,buffer,strlen(buffer),0);  /* sends to server via socket     */

    memset(buffer,0,256);                  /* clears buffer                  */
    recv(sockfd,buffer,255,0);             /* reads response from server     */
    printf("%s\n",buffer);                 /* displays locally               */

    closesocket(sockfd);

    return 0;
}
```

Next week…

Good news: no quiz!

Bad news: Midterm Exam!

worth 20% of final grade

designed for 1-1½ hours, will allow 2 hours

types of questions:

- true/false
- discussion
- understand/modify code

Advice: study lecture notes, text, quizzes, homeworks
See review sheet online