Мрежово програмиране

Програмиране на приложно ниво – III част

доц. д-р Йордан Денев
denev@fmi.uni-sofia.bg

Java интерфейси

- Създаване на клиентски socket

```
Socket MyClient;
try {
  MyClient = new Socket("Machine name", PortNumber);
}
catch (IOException e)
  { System.out.println(e); }
```

❑ Създаване на входен и изходен поток

```
DataInputStream input;
try {
input = new
   DataInputStream(MyClient.getInputStream());
}
catch (IOException e) { System.out.println(e); }


PrintStream output;
try {
output = new
   PrintStream(MyClient.getOutputStream());
}
catch (IOException e) { System.out.println(e); }
```

- Някои методи на входен поток

`int read(byte[] b)`

Reads some number of bytes from the contained input stream and stores them into the buffer array b. The number of bytes read is, at most, equal to the length of b

`int read(byte[] b, int off, int len)`

Reads up to **len** bytes of data from the contained input stream into an array of bytes.

- Някои методи на изходен поток

`void write(byte[] buf, int off, int len)`

Write **len** bytes from the specified byte array starting at offset **off** to this stream.

`void write(int b)`

Write the specified byte to this stream.

Затваряне на връзката

```
try {
    output.close();
    input.close();
    MyClient.close();
}
catch (IOException e) { System.out.println(e); }
```

❑ Създаване на сървърен socket

```
ServerSocket MyService;
try  {
   MyServerice = new ServerSocket(PortNumber);
}
catch (IOException e) { System.out.println(e); }

Socket clientSocket = null;
try {
   clientSocket = MyService.accept();
}
catch (IOException e) { System.out.println(e); }
```

❑ Създаване на входен и изходен поток

```
DataInputStream input;
try  {
   input = new
   DataInputStream(clientSocket.getInputStream());
}
catch (IOException e) { System.out.println(e); }


PrintStream output;
try {
   output = new
   PrintStream(clientSocket.getOutputStream());
}
catch (IOException e) { System.out.println(e); }
```

- Затваряне на връзката

```
try {
   output.close();
   input.close();
   serviceSocket.close();
MyService.close();
}
catch (IOException e) { System.out.println(e); }
```

- Примери –> `server.java, client.java`

Winsock интерфейс

- Аналогичен на стандартния socket интерфейс.

```
SOCKET  WSAAPI socket(
   __in int af, /* Network */
   __in int type, /* TCP or UPD,..*/
   __in int protocol );

int bind(
   __in SOCKET s,
   __in const struct sockaddr *name,
   __in int namelen );
```

```
int listen(
   __in SOCKET s,
   __in int backlog )

SOCKET accept(
   __in SOCKET s,
   __out struct sockaddr *addr,
   __inout int *addrlen );

int connect(
   __in SOCKET s,
   __in const struct sockaddr *name,
   __in int namelen )
```

- Примери –> winsock_intro.html, winsock_server.html, winsock_client.html

UDP протокол

UPD е дейтаграмен протокол, който е

- несигурен, т.е. не се гарантира получаването на изпратените дейтаграми; гарантира се само целостта на дейтаграмата (ако се плучи).
- не гарантира, че реда на получаване е същият на изпращане;
- с по висока производителност, отколкото TCP.

UDP програмиране

- Creating UDP sockets.
  - ✓Client
  - ✓Server
- Sending data.
- Receiving data.
- Connected Mode.

UDP socket

```
int socket(int family,int type,int proto);

int sock;
sock = socket( AF_INET,
               SOCK_DGRAM,
               0);
if (sock<0) { /* ERROR */ }
```

Binding to well known address (typically done by server only)

```
int mysock;
struct sockaddr_in myaddr;

  mysock = socket(AF_INET,SOCK_DGRAM,0);
  myaddr.sin_family = AF_INET;
  myaddr.sin_port = htons( 1234 );
  myaddr.sin_addr = htonl( INADDR_ANY );
  bind(mysock, &myaddr, sizeof(myaddr));
```

Sending UDP Datagrams

```
ssize_t sendto(   int sockfd,
                  void *buff,
                  size_t nbytes,
                  int flags,
                  const struct sockaddr* to,
                  socklen_t addrlen);
```

sockfd is a UDP socket

buff is the address of the data (**nbytes** long)

to is the address of a sockaddr containing the destination address.

Return value is the number of bytes sent, or -1 if error.

Receiving UDP Datagrams

```
ssize_t recvfrom(   int sockfd,
                    void *buff,
                    size_t nbytes,
                    int flags,
                    struct sockaddr* from,
                    socklen_t *fromaddrlen);
```

sockfd is a UDP socket

buff is the address of a buffer (**nbytes** long)

from is the address of a sockaddr.

Return value is the number of bytes received and put into buff, or -1 on error.

- If buff is not large enough, any extra data is lost forever...
- You can receive 0 bytes of data!
- The **sockaddr** at **from** is filled in with the address of the sender.
- You should set **fromaddrlen** before calling.
- If **from** and **fromaddrlen** are NULL we don't find out who sent the data.

- The return value of **sendto()** indicateshow much data was accepted by the O.S. for sending as a datagram – not how much data made it to the destination.
- There is no error condition that indicates that the destination did not get the data!!!
- Unless you do something special **recvfrom** doesn’t return until there is a datagram available.

Typical UDP client code

1. Create UDP socket.
2. Create **sockaddr** with address of server.
3. Call **sendto(),** sending request to the server. **No call to bind() is necessary!**
4. Possibly call **recvfrom()** (if we need a reply).

Typical UDP Server code

1. Create UDP socket and bind to well known address.
2. Call **recvfrom()** to get a request, nothing the address of the client.
3. Process request and send reply back with **sendto().**

UDP Echo Server

```
int mysock;
struct sockaddr_in myaddr, cliaddr;
char msgbuf[MAXLEN];
socklen_t clilen;
int msglen;
  mysock = socket(PF_INET,SOCK_DGRAM,0);
  myaddr.sin_family = AF_INET;
  myaddr.sin_port = htons( S_PORT );
  myaddr.sin_addr = htonl( INADDR_ANY );
  bind(mysock, &myaddr, sizeof(myaddr));
  while (1) {
      len=sizeof(cliaddr);
      msglen=recvfrom(mysock,msgbuf,MAXLEN,0,
            cliaddr, &clilen);
      sendto(mysock,msgbuf,msglen,0,cliaddr,clilen);
  }
```

Connect in UDP

- A UDP socket can be used in a call to **connect().**
- This simply tells the O.S. the address of the peer.
- No handshake is made to establish that the peer exists.
- No data of any kind is sent on the network as a result of calling **connect()** on a UDP socket.

Once a UDP socket is *connected:*

- can use **sendto()** with a null dest.
- address
- can use **write()** and **send()**
- can use **read()** and **recv()**

Only datagrams from the peer will be returned.