

**CS 580 Client-Server Programming**  
**Spring Semester, 2005**  
**Doc 6 Server Intro**  
**Contents**

What is a Server? .....	3
Sockets .....	6
Java TCP Sockets .....	7
A Simple Date Server .....	8
Simple Server Issues .....	12
Backlog .....	13
Multi-homed Machines .....	15
Reusing a Port .....	16
End of Line .....	17
End of File .....	19
End of Message .....	20
Buffers .....	22

**Copyright** ©, All rights reserved. 2005 SDSU & Roger Whitney,  
5500 Campanile Drive, San Diego, CA 92182-7700 USA.  
OpenContent (<http://www.opencontent.org/opl.shtml>) license  
defines the copyright on this document.

## References

Java Network Programming, Harold

VisualWorks Internet Client Developer's Guide

java.net.ServerSocket & Socket. See

<http://java.sun.com/j2se/1.4.2/docs/api/>

## Reading Assignment

### Java

Java Network Programming, Harold

Basic Network Concepts, Chapter 2

If you are not familiar with basic networking

Java I/O Chapter 4

If you are weak on Streams, Readers and Writers

Sockets for Servers Chapter 10

# What is a Server?

## Server

- Any program that waits for incoming communication requests from a client
- Extracts requested information from data and return to client

Basic algorithm:

```
while (true) {  
    Wait for an incoming request;  
    Perform whatever actions are requested;  
}
```

## Example - Echo Server

```
| server |  
server := SocketAccessor newTCPserverAtPort: 9009.  
server listenFor: 5.  
  
[ | acceptedSocket |  
  "wait for a new connection"  
  acceptedSocket := server accept.  
  
  "fork off processing of the new stream socket"  
  [ | stream char |  
    stream := acceptedSocket readAppendStream.  
    stream lineEndTransparent.  
    [ (char := stream next) isNil ] whileFalse: [  
      stream nextPut: char; commit ].  
    stream close.  
  ] forkAt: Processor userSchedulingPriority -1.  
] repeat.
```

## **Some Basic Server Issues**

- How to wait for an incoming request?
- How to know when there is a request?
- What happens when there are multiple requests?
- How do clients know how to contact server?
- How to parse client request?
- How do we know when the server has the entire request?

## **Sockets**

### **Streams verses Buffers**

Both Java & Smalltalk provide access to socket data via

- Streams
- Buffers

Stream access is easier

Buffer access can be faster

We will cover Stream access first

## **Java TCP Sockets**

### **Main Classes**

#### **ServerSocket**

Used by servers to listen for clients

#### **Socket**

Used by clients to talk to servers

Used by servers to talk to clients

#### **ServerSocket basic methods**

```
public ServerSocket(int port) //port = 0 gives random port
public ServerSocket(int port, int backlog)
public ServerSocket(int port, int backlog, InetAddress bindAddress)
```

```
public Socket accept() throws IOException
public void close() throws IOException
public int getLocalPort()
```

#### **Socket basic methods**

```
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
```

## A Simple Date Server

```
public class DateServer {  
    private static Logger log = Logger.getLogger("dateLogger");  
  
    public static void main (String args[]) throws IOException {  
        ProgramProperties flags = new ProgramProperties( args);  
        int port = flags.getInt( "port" , 8765);  
        new DateServer().run(port);  
    }  
  
    public void run(int port) throws IOException {  
        ServerSocket input = new ServerSocket( port );  
        log.info("Server running on port " + input.getLocalPort());  
  
        while (true) {  
            Socket client = input.accept();  
            log.info("Request from " + client.getInetAddress());  
            processRequest(  
                client.getInputStream(),  
                client.getOutputStream());  
            client.close();  
        }  
    }  
}
```



## Processing Client Request

```
void processRequest(InputStream in,OutputStream out)
    throws IOException {

    BufferedReader parsedInput =
        new BufferedReader(new InputStreamReader(in));

    boolean autoflushOn = true;
    PrintWriter parsedOutput = new PrintWriter(out,autoflushOn);

    String inputLine = parsedInput.readLine();

    if (inputLine.startsWith("date")) {
        Date now = new Date();
        parsedOutput.println(now.toString());
    }
}
```

Note: This server is just a first example. It needs a lot of work. We will be working on improving it in later lectures.

## Running the Server

Sample run of DateServer.

(I typed everything appearing in bold font here.)

rohan 16-> <b>java -jar DateServer.jar</b> Feb 19, 2004 10:56:59 AM DateServer run INFO: Server running on port 8765	
	Al 13-> <b>telnet rohan.sdsu.edu 8765</b> Trying 130.191.3.100... Connected to rohan.sdsu.edu. Escape character is '^['.
Feb 19, 2004 10:57:34 AM DateServer run INFO: Request from /68.7.92.191	
	<b>date</b> Thu Feb 19 10:57:39 PST 2004 Connection closed by foreign host.

## **Warning About telnet Usage**

Using telnet to interact with a server is

- Useful as development tool
- Useful in debugging
- Not for end-users

Don't design server assuming it interacts directly with a human!

## Simple Server Issues

- How do we test our server?
- Request processing blocks any other connections.

Using our SimpleDateServer

Client A builds connection to server,  
Client A goes to lunch  
Client B builds connection to server and ... :-(

Solution:

Multiple connections need to be accepted concurrently.

## Backlog

TCP accepts connections before the server is ready

TCP keeps a backlog queue of connections server has not accepted

Java ServerSocket constructor

- Allows you to request a maximum size of the backlog queue
- The OS will not exceed its maximum TCP backlog queue size
- The OS silently reduces your request to its maximum

There is no reasonable way to find out:

- OS's maximum TCP backlog queue size

## **Backlog Experiment**

Start the SimpleDateServer

Connect to the server using telnet

While the server is waiting for you to type something

Connect to the server with a second telnet session

In the second session type “date” and return

What happens?

Now type “date” and return in the first session

What happens?

## Multi-homed Machines

Some machines have two or more physical network interface

- Rohan has two ethernet cards
- Some laptops have ethernet & 802.11a

Each network interface has its own IP address

```
public ServerSocket(int port)
```

- Listens to the give port on all IP addresses for the machine

```
public ServerSocket(int port, int backlog, InetAddress bindAddress)
```

- Listens to the give port only on the one IP address

## Reusing a Port

Closing TCP connections can remain for several minutes

TCP may block use of the port until the connection is gone

This can be annoying in development

ServerSocket method `setReuseAddress(boolean on)`

- Allows the port to be reused



## End of Line

Platform	End of Line Convention
Unix	Line Feed (LF)
Macintosh OS 9	Carriage return (CR)
Windows	CR-LF

LF is ASCII character 10

CR is ASCII character 13

A server should not make assumptions about a client's platform

A client should not make assumptions about a server's platform

Client-server protocol should specify which characters are used

## End of Line & Smalltalk

Java and Smalltalk programs run on all major platforms

Smalltalk assumes files use platform's end of line convention

Smalltalk input streams convert platforms end of line to CR

Smalltalk output streams convert CR to platform's end of line

This makes writing cross platform programs easier

Don't want this to happen socket streams

`BufferedExternalStream>> lineEndTransparent`

- Turns off conversion between CR & end of line

How does Java handle this?

## **End of File**

On a stream connected to a socket

End of file indicates that the connection has been closed!

Don't use end of file to determine when other end is done talking!

## End of Message

How do we know when we are at the end of a message?

```
BufferedReader parsedInput =  
    new BufferedReader(new InputStreamReader(in));
```

```
char[] message = new char[500];  
int sizeRead = parsedInput.read(message, 0, 500);
```

If

- client's message is less than 500 characters and
- Above read returns

We still may not have the entire message!

Why?

## **End of Message**

A good client-server protocol specifies

- How to determine the end of a message

Main methods used:

- Include the length of the message
- Specify end of message character sequence

## Buffers

Java & Smalltalk streams are buffered

TCP buffers output before sending

A server cannot read bytes left in a client's buffer

`PrintWriter flush();`

- Sends data in stream to OS