CS 580 Client-Server Programming
Spring Semester, 2004
Doc 15 Socket Options & Server Types

Socket Options .................................................................................................................. 2
Timeouts .............................................................................................................................. 5
Buffer Size .......................................................................................................................... 6
Nagle’s Algorithm ................................................................................................................ 13
Linger on close .................................................................................................................... 15
Keep Alive .......................................................................................................................... 16
Urgent (Out of Band) Data ................................................................................................ 17
Types of Servers ................................................................................................................ 18
Iterative verses Concurrent ............................................................................................... 19
Stateless verses Stateful Servers ...................................................................................... 24
Modes of Operation ........................................................................................................... 25

References

Java On-line API http://java.sun.com/j2se/1.4.1/docs/api/

VisualWorks Internet Client Developer’s Guide, Socket Programming Chapter 2,
docs/NetClientDevGuide.pdf in VW 7 distribution

VisualWorks 7 Source code


TCP/IP Illustrated Vol 1, Stevens, 1994, chapter 20.

Internetworking with TCP/IP, BSD Socket Version Vol. 3, Comer, Stevens,
Prentice-Hall, 1993

Copyright ©, All rights reserved. 2004 SDSU & Roger Whitney, 5500 Campanile
Drive, San Diego, CA 92182-7700 USA. OpenContent
(http://www.opencontent.org/pl.shtml) license defines the copyright on this document.
Socket Options

- Timeouts
- Buffer Size
- Multi-Homing
- No Delay for small data
- Linger on close
- Keep-Alive
- Urgent-Data
Setting Socket Options - VisualWorks

One method to set standard TCP socket options

Uses Berkley sockets naming conventions

SocketAccessor>> setOptionsLevel: SocketAccessor SOL_SOCKET
    name: optionToSet
    value: optionValue.

setOptionsLevel: is always set to SocketAccessor SOL_SOCKET

Values for name: are found in SocketAccessor class under constants-
socket options protocol

Example

| serverSocket |
serverSocket := SocketAccessor newTCPserverAtPort: 4444.
serverSocket
    setOptionsLevel: SocketAccessor SOL_SOCKET
    name: SocketAccessor SO_RCVBUF
    value: 56 * 1024.
Reading Current Values

SocketAccessor>> getOptionsLevel: SocketAccessor SOL_SOCKET
    name: option

Returns result in a byte array

To interpret the bytes

• Change the result’s class to UninterpretedBytes
• Convert to the type expected

| childSocket rawBytes bufferSize |
childSocket := SocketAccessor
    newTCPclientToHost: 'rugby.sdsu.edu'
    port: 8008.
rawBytes := childSocket
    getOptionsLevel: SocketAccessor SOL_SOCKET
    name: SocketAccessor SO_RCVBUF.
rawBytes changeClassTo: UninterpretedBytes.
bufferSize := rawBytes longAt: 1.
^bufferSize
Timeouts

Socket will time out after specified time of inactivity

Java JDK 1.4 and later

Both Socket and ServerSocket class support:

void setSoTimeout(int timeoutInMilliseconds) throws SocketException
void getSoTimeout() throws SocketException

Must be sent before performing a read

Read throws SocketTimeoutException when socket times out

Not normally used on ServerSockets

VisualWorks

Does not support SO_RCVTIMEO or SO_SNDTIMEO

Need to use non-stream access for timeouts
Buffer Size

Each TCP socket has

- Receive buffer
- Send Buffer

Buffers are in the TCP stack space (not the VM)

Buffer size should:

- Be at least 16KB on Ethernet
- Applications that send lots of data use 48KB or 64KB

TCP does not allow the sender to overflow the receiver’s buffer

So the receiver’s receive buffer as large as the sender’s send buffer

Buffers larger than 64KB require special set up
Default Buffer Size

Depends on platform

Has changed over time

<table>
<thead>
<tr>
<th>OS</th>
<th>Receive buffer Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>32KB</td>
</tr>
<tr>
<td>Mac OS 10</td>
<td>32KB+ (33304 bytes)</td>
</tr>
</tbody>
</table>
Setting the Buffer Size – VisualWorks

Setting the Receive buffer

aSocketAccesor setOptionsLevel: SocketAccessor SOL_SOCKET
   name: SocketAccessor SO_RCVBUF
   value: newBufferSize.

Setting the Send buffer

aSocketAccesor setOptionsLevel: SocketAccessor SOL_SOCKET
   name: SocketAccessor SO_SNDBUF
   value: newBufferSize.
Setting the Buffer Size – Java

```java
void setReceiveBufferSize(int size) throws SocketException
int getReceiveBufferSize() throws SocketException

void setSendBufferSize(int size) throws SocketException
int getSendBufferSize() throws SocketException
```

A Socket object has both a send & receive buffer

A ServerSocket only has a receive buffer
Java Example

In this example the default buffer size will be fine

Setting the buffer size just to show how to do it

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

public class ServerWithTimeout extends Thread {
    static final int CLIENT_TIMEOUT = 3 * 1000; // in milliseconds
    static final int BUFFER_SIZE = 16 * 1024;
    ServerSocket acceptor;

    public static void main(String[] args) throws IOException {
        int port = Integer.parseInt( args[1]);

        ServerWithTimeout server = new ServerWithTimeout( port );
        server.start();
    }

    public ServerWithTimeout(int port ) throws IOException {
        acceptor = new ServerSocket(port);
        acceptor.setReceiveBufferSize( BUFFER_SIZE );
    }
```
Java Example Continued

public void run()
{
    while (true)
    {
        try
        {
            Socket client = acceptor.accept();
            processRequest( client );
        }
        catch (IOException acceptError)
        {
            // for a later lecture
        }
    }
}

void processRequest( Socket  client) throws IOException
{
    try
    {
        client.setReceiveBufferSize( BUFFER_SIZE);
        client.setSoTimeout( CLIENT_TIMEOUT);
        processRequest( client.getInputStream(),
                        client.getOutputStream());
    }
    finally
    {
        client.close();
    }
}
Java Example Continued

void processRequest(InputStream in, OutputStream out) throws IOException
{
    BufferedReader parsedInput = null;
    PrintWriter parsedOutput = null;
    try
    {
        parsedInput = new BufferedReader(new InputStreamReader(in));
        parsedOutput = new PrintWriter(out, true);

        String inputLine = parsedInput.readLine();

        if (inputLine.startsWith("date"))
        {
            Date now = new Date();
            parsedOutput.println(now.toString());
        }
    }
    catch (SocketTimeoutException clientTooSlow)
    {
        parsedOutput.println("Connection timed out");
    }
}
}
Nagle’s Algorithm

Delays transmission of new TCP packets while any data remains unacknowledged

Allows TCP to merge data into larger packets before sending

Introduced to avoid lots of small packets across a WAN

Delay is on be default
Java

class Socket
{
    void setTcpNoDelay(Boolean noDelay) throws SocketException
    void getTcpNoDelay() throws SocketException
}

Set noDelay to true to turn delay off

VisualWorks

aSocketAccesor setOptionsLevel: SocketAccessor SOL_SOCKET
    name: SocketAccessor TCP_NODELAY
    value: delayValue.

Set delayValue to
• 1 to turn delay off
• 0 to turn delay on

or use the shorter method

aSocketAccesor tcpNoDelay: noDelay
Linger on close

Determines what happens when a socket is closed

How long does the socket remain after close to

• Acknowledge packets
• Retransmit lost packets

Default is to

• Allow the application to continue
• TCP handles sending unsent data & rejecting new requests
Keep Alive

Send packet on inactive connection to prevent timeouts

At least 2 hour delay between sending keep alive packets

Long delay limits its usefulness
Urgent (Out of Band) Data

Urgent data can be read out of order

• Read before data that was sent before it

Java 1.4

• Supports sending of urgent data
• Does not promote urgent data in the input stream
Types of Servers

- Connectionless (UDP) verse Connection-Oriented (TCP)
- Iterative verses Concurrent
- Stateless verse stateful
Iterative verses Concurrent

Iterative

Single process

Handles requests one at a time

Good for low volume & requests that are answered quickly
Iterative Example

import java.net.Socket;
import java.net.ServerSocket;
import java.io.*;
import java.util.Date;

class SimpleDateServer {

    public static void main(String[] args) throws IOException {
        ServerSocket acceptor = new ServerSocket(4567);

        while (true) {
            Socket client = acceptor.accept();
            processRequest(
                client.getInputStream(),
                client.getOutputStream());
            client.close();
        }
    }

    static void processRequest(InputStream in, OutputStream out) throws IOException {
        BufferedReader parsedInput =
            new BufferedReader(new InputStreamReader(in));
        PrintWriter parsedOutput = new PrintWriter(out, true);
        parsedOutput.println(now.toString());
    }
}
Concurrent

Handle multiple requests concurrently

Normally uses thread/processes

Needed for high volume & complex requests

Harder to implement than iterative

Must deal with currency
Concurrent Server Example

| 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.
Single Process/Thread Concurrent Server

One can implement a concurrent server using one thread/process

while (true) {
    check if any new connects (non-block accept)
    if new connection accept
    process a little on each current request
}
Stateless verses Stateful Servers

State information

• Information maintained by server about ongoing interactions with clients

Stateless server

• Server that does not maintain state information

Stateful server

• Server that does maintain state information

State information cause problems

• Consumes resources

• How long does one maintain the state?
Modes of Operation

Stateful servers sometimes have different modes of operation

Each mode has a set of legal commands

In Login mode only the commands password & username are acceptable

After successful login client-server connection in transaction mode

In transaction mode commands X, Y Z are legal

These modes are also called server states or just states