References

Java On-line API http://java.sun.com/j2se/1.5.0/docs/api/index.html


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

Server - Basic Algorithm

while (true) {
    Wait for an incoming request;
    Perform whatever actions are requested;
}
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();
    }
}
void processRequest(InputStream in, OutputStream out) throws IOException {
    Read client request from in
    Determine response
    Send response back on out
}
Issues

Socket Issues

Configuring Server

Logging

Handling Multiple Requests

Interacting with Database

Structuring Server
Socket Options

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

Socket will time out after specified time of inactivity

Java

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
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
Backlog

Server socket has queue for incoming requests

Can set the size of the queue for the server socket

Can not exceed OS limits

Default value is 50 in 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;
    static final int CONNECTION_QUEUE_SIZE = 40;
    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, CONNECTION_QUEUE_SIZE);
        acceptor.setReceiveBufferSize( BUFFER_SIZE );
    }
}
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

```java
void processRequest(InputStream in, OutputStream out) throws IOException {
    Process client request
    
    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 by default

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

Determines what happens when a socket is closed

How long does the socket remain after being closed

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

Supports sending of urgent data

Does not promote urgent data in the input stream
Application Parameters & Configuration Files
Applications normally have configuration files to store

- User preferences
- Cached values
- Window settings
- Port numbers
- Database connection information
- Log file information
- Recent documents/web pages
- Cookies
- Values that need changing without recompiling
Environment Variables & Command line

cvs co assignment2
ls -la
ps -aux
Servers Config files & Command line flags

Servers normally use configuration files & command line flags

Environment variables are not used much in servers (why?)
Java & Config files

Some systems have libraries to handle config files & command line arguments

JDK does not seem to have such classes

There are be a number of Java libraries that provide such support
   Apache Commons configuration (http://commons.apache.org/configuration/)

sdsu Java library is one such library
sdsu.util.ProgramProperties

Parses
Configuration files
Command line arguments

Command Line argument
-flag=value
flag value
-flag
--xyz
-- (ignore rest of the command line)

File Formats
properties format

# A comment to the end of the line
key1=value1
key2=value2   with spaces
key3 with spaces=value3 # part of the value

sdsu.util.LabeledData format

# A comment to the end of the line,
key1 = value1;
key2='value2   with spaces';
'key3 with spaces'=value3;       # a comment
import sdsu.util.ProgramProperties;

public class ConfigurationExample {
    public static void main(String args[]) {
        try {
            ProgramProperties flags =
                new ProgramProperties( args, "configurationFile");
            String nameValue =
                flags.getString( "name" , "No name given");
            int size = flags.getInt( "size", 0);
            boolean switchOn = flags.containsKey( "s");
            System.out.println( " nameValue: " + nameValue);
            System.out.println( " size: " + size);
            System.out.println( " switchOn: " + switchOn);
        }
        catch (java.io.IOException readParseProblem) {
            System.err.println( "Program aborted on error " +
                readParseProblem);
        }
    }
}

Example
Sample Runs

java ConfigurationExample

Output
nameValue: Roger
size: 12
switchOn: false

java ConfigurationExample -s -name Pete

Output
nameValue: Pete
size: 12
switchOn: true

java ConfigurationExample -conf=otherFile

Output
nameValue: Sam
size: 8
switchOn: true
require 'optparse'

class SampleOptionParser
  def initialize
    parseOptions(ARGV)
  end

  def parseOptions(args)
    options = OptionParser.new
    options.on("-x")  { |value| @x = true }
    options.on("-s SIZE", "--size SIZE", Integer, "Size of new file in bytes")  { |size| @fileSize = size }
    options.on("-p=[PORT]", "--port=[PORT]", Integer, 
               "Port for server")  { |port| @fileSize = size }
    options.on_tail("-h", "--help", "Show this message") do
      puts options.to_s
      exit
    end
    options.on_tail("--version", "Show version") do
      puts OptionParser::Version.join(".")
      exit
    end

    options.parse(args)
  end
end
Ruby Example

Al 77->ruby SampleOptionParser.rb --h

Usage: SampleOptionParser [options]

- x
- s, --size SIZE                   Size of new file in bytes
- p, --port=[PORT]                 Port for server
- h, --help                        Show this message
   --version                        Show version
Logging
Logging

Performance tuning
Upgrade justification
Problem tracking
Access counting
What should be logged?

Date and time
Service that caused the entry
Client address that caused the entry
Host on which the server runs
Event

Apache Access Log
211.90.88.43 - - [21/Oct/2002:08:33:29 -0700] "GET /scripts/..%25%35%63../winnt/system32/cmd.exe?/c+dir HTTP/1.0" 404 303
211.90.88.43 - - [21/Oct/2002:08:33:30 -0700] "GET /scripts/..%252f../winnt/system32/cmd.exe?/c+dir HTTP/1.0" 404 303

Apache Error Log
[Mon Oct 21 08:33:29 2002] [error] [client 211.90.88.43] File does not exist: /opt/etc/apache-1.3.26/htdocs/scripts/..%5c../winnt/system32/cmd.exe
[Mon Oct 21 08:33:30 2002] [error] [client 211.90.88.43] File does not exist: /opt/etc/apache-1.3.26/htdocs/scripts/..%2f../winnt/system32/cmd.exe
Java Logging

Multiple log levels
Multiple output formats
Output to different IO devices

Filters for additional filtering of message to accept
ResourceBundles for localization of log messages
Initialization of loggers by configuration file
Hierarchical loggers in one program

Output formats
XML (default for files output)
Normal Text (default for screen output)

Output devices
Stream
System.err
File or rotating set of files
Socket for network logging
Memory

Log Levels
ALL
SEVERE (highest value)
WARNING
INFO (usual default)
CONFIG
FINE
FINER
FINEST (lowest value)
OFF
Example

```java
import java.util.logging.*;
public class SimpleLoggingExample {
    private static Logger logger = Logger.getLogger("edu.sdsu.cs580");

    public static void main (String args[]) {
        new SimpleLoggingExample().someLogMessages();
    }

    public void someLogMessages() {
        logger.severe("A severe log message");
        Logger.getLogger("edu.sdsu.cs580").fine("A fine message");
        logger.warning("Be careful");
    }
}
```

Output To System.err

Feb 16, 2004 10:51:37 PM Logging someLogMessages WARNING: Be careful

Default Settings

Use a ConsoleHandler
Level set to INFO
System administrator can change default settings
Logging Messages

Convenience Methods

severe( String message);
warning( String message);
info( String message);
config( String message);
fine( String message);
finer( String message);
finest( String message);

Convenience Methods for Tracing Methods

entering(String sourceClass, String sourceMethod);
entering(String sourceClass, String sourceMethod, Object parameter);
entering(String sourceClass, String sourceMethod, Object[] parameters);
exiting(String sourceClass, String sourceMethod);
exiting(String sourceClass, String sourceMethod, Object result);

Log Methods

log(Level logLevel, String message);
log(Level logLevel, String message, Object parameter);
log(Level logLevel, String message, Object[] parameters);
log(Level logLevel, String message, Throwable exception);
public class MessageTypes {
    private static Logger logger = Logger.getLogger("edu.sdsu.cs580");

    static {
        try {
            Handler textLog = new FileHandler("textLog.txt");
            textLog.setFormatter( new SimpleFormatter());
            textLog.setLevel(Level.ALL);
            Handler xmlLog = new FileHandler("xmlLog.txt");
            xmlLog.setFormatter( new XMLFormatter());
            xmlLog.setLevel(Level.ALL);
            logger.addHandler(textLog);
            logger.addHandler(xmlLog);
            logger.setLevel(Level.ALL);
        }
        catch (IOException fileError) {
            System.err.println("Could not open log files");
        }
    }
}
public static void main (String args[]) {
    new MessageTypes().someLogMessages();
}

public void someLogMessages() {
    logger.entering("MessageTypes", "someLogMessages");
    Vector data = new Vector();
    data.add("Cat");
    logger.log(Level.SEVERE, "Show Vector", data);
    logger.severe("A severe log message");
    logger.logp(Level.SEVERE, "MessageTypes", "someLogMessages", "Logp example");
    try {
        int zeroDivide = 1 / (1 - 1);
    }
    catch (Exception zeroDivide) {
        logger.log(Level.SEVERE, "Exception example", zeroDivide);
    }
    logger.exiting("MessageTypes", "someLogMessages");
}
Sample Output

SimpleFormatter Output
Feb 16, 2004 11:01:53 PM MessageTypes someLogMessages FINER: ENTRY
Feb 16, 2004 11:01:53 PM MessageTypes someLogMessages SEVERE: Show Vector
Feb 16, 2004 11:01:53 PM MessageTypes someLogMessages SEVERE: A severe log message
Feb 16, 2004 11:01:54 PM MessageTypes someLogMessages SEVERE: Logp example
Feb 16, 2004 11:01:54 PM MessageTypes someLogMessages SEVERE: Exception example
java.lang.ArithmeticException: / by zero
  at MessageTypes.someLogMessages(MessageTypes.java:45)
  at MessageTypes.main(MessageTypes.java:32)
Feb 16, 2004 11:01:54 PM MessageTypes someLogMessages FINER: RETURN

XMLFormatter Sample Output
<?xml version="1.0" encoding="US-ASCII" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
  <record>
    <date>2004-02-16T23:01:53</date>
    <millis>1077001313695</millis>
    <sequence>0</sequence>
    <logger>edu.sdsu.cs580</logger>
    <level>FINER</level>
    <class>MessageTypes</class>
    <method>someLogMessages</method>
  </record>
</log>
FileHandlers

Can be set to rotate files

Can be located in temp directory

Can be set to
  Append existing files
  Overwrite existing files (default)

To change append setting either

Use constructor

    FileHandler(String pattern, boolean append)

Or use configuration file
Loggers

Can have
  Multiple handlers
  Multiple handlers of same type

Loggers and handlers have differ log levels

Logger
  Drops all messages below it log level
  Passes remaining messages to all handlers
  Handler can further drop more messages
Logger Names

Logger names are arbitrary
- `Logger.getLogger("edu.sdsu.cs580")`
- `Logger.getLogger("foo")`
- `Logger.getLogger(""")`

Sun recommends using hierarchical names with format
- "domain.package"
- "domain.package.class"

Loggers inherit settings from “parent” logger

Logger "edu.sdsu.cs580" would inherit settings of "edu.sdsu"
Logger Scope

Logger settings can be defined in
   Program
   Configuration File

Logger settings defined in a program exist only in that program

Logger settings defined in a configuration file can be used by multiple programs
Sample Configuration File

# Use two loggers
handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler

# Default global logging level.
.level= WARNING

# File logger default settings
# Default file output is in user's home directory (%h/).
# %g – use generation numbers to distinguish rotated logs
# limit = max size of each log file
# count = number of output files to cycle through
java.util.logging.FileHandler.pattern = %h/cs580Server%g.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 3
java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter

# Limit the message that are printed on the console to INFO and above.
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

# Set levels of specific loggers
edu.sdsu.level = SEVERE
edu.sdsu.cs580.level = INFO
Using the Configuration File

Assume that configuration file is in
   Local directory
   In a file called cs580Log.properties

The following command will use the configuration file

   java -Djava.util.logging.config.file=cs580Log.properties  yourClassGoesHere