Курсов проект № 1 по “Интернет програмиране с Java”

Проект 1.1 – UDP Chat Client/Server:

Да се напише клиент/сървър приложение на Java, което използвайки възможностите на пакета java.net дава възможност за комуникация между много потребители в локална мрежа. Изпращането и приемането на съобщения трябва да бъде реализирано с UDP сокети. Трябва да се напишат 2 програми – UDP Chat сървър и UDP Chat клиент. UDP Chat сървърът трябва да служи като централизирана точка, през която минават съобщенията, които са предназначени за всички потребители. UDP Chat клиентът трябва да е клиентско приложение, което си комуникира със UDP Chat сървъра и другите UDP Chat клиенти чрез UDP пакети.

UDP Chat клиентът представлява програма, която приема потребителски команди и ги изпълнява и същевременно приема изпратени до нея съобщения и ги отпечатва. Всеки потребител трябва да може да изпраща съобщения до всички останали или до конкретен друг потребител. Получаването и изпращането на съобщения трябва да става по UDP на порт 2002. Трябва да се реализират следните команди:

SEND IP_ADDRESS MESSAGE – за изпращане на съобщение до конкретен потребител. Съобщението се изпраща чрез UDP пакет директно до зададения IP адрес на порт 2002 (не минава през сървъра).

SEND * MESSAGE – за изпращане на съобщение до всички потребители. Тази команда, без изменение се изпраща чрез UDP пакет до UDP Chat сървъра на порт 2002. IP адресът на UDP Chat сървъра е предварително зададен като константа в клиентското приложение.

QUIT – за изход от клиентската програма

При пристигане на съобщение от някой потребител то трябва да се извежда по подходящ начин. При стартиране на клиента той трябва да изпраща до UDP Chat сървъра команда “REGISTER ”, с което го уведомява, че се е включил в Chat-а. При изход, клиентът трябва да изпраща друга команда “UNREGISTER ” до UDP Chat сървъра, за да го уведоми, че е излязъл от Chat-а. Във всеки един момент потребителят трябва да има възможност да изпълнява произволна от изброените по-горе команди.

UDP Chat сървърът представлява програма, която слуша на UDP порт 2002, приема команди и да ги изпълнява по подходящ начин. Командите са следните:

REGISTER – добавя IP адреса в списъка от клиентите (ако го няма вече там)

UNREGISTER – премахва IP адреса от списъка от клиентите (ако го има там)

SEND * MESSAGE – изпраща зададеното съобщение до всички клиенти чрез UDP пакет на порт 2002, като включва в него и IP адреса на изпращача, т.е. на този, от когото е получено, за да знае клиентът кой го е изпратил първоначално.

При пристигане на друга команда, тя се игнорира. UPD Chat сървърът трябва да поддържа списък от IP адресите на клиентите си, като при стартиране този списък е празен. Сървърът трябва да може да изпълнява много команди от различни потребители, дори ако са изпратени едновременно.

Пример: Нека UDP Chat сървърът е стартиран на адрес 192.168.0.31 и слуша на порт 2002. Alan и Bob са потребители в локална мрежа съответно с IP адреси 192.168.0.1 и 192.168.0.2. Двамата стартират вашия UDP Chat клиент. Съответно сървърът получава команди за регистрация на IP адреси 192.168.0.1 и 192.168.0.2.

Потребител Alan пише командите:

SEND * Hi all!

SEND 192.168.0.2 Hi Bob!

Първата команда се изпраща без изменение до сървъра (по UDP към 192.168.0.31:2002), а втората просто предизвиква изпращане на клиента 192.168.0.2 по UDP на порт 2002 съобщението “Hi Bob!”.

След това потребител Bob пише командите:

SEND 192.168.0.1 Hi Alan, I am pleased to chat with you!

SEND * Anybody for Counter Strike?

Те се обработва отново съгласно описанието на клиентското приложение.

След това Alan пише командите:

SEND 192.168.0.2 Bob, I don’t like this game. I have to work now.

SEND 192.168.0.2 Goodbye Bob.

SEND * Goodbye all.

QUIT

След това потребител Bob пише командите:

SEND * Ok. I am leaving. The game session will start at 11.30 pm. Bye all.

QUIT

Сървърът приема всички команди от вида “SEND * MESSAGE“ и разпраща съобщенията до всички от списъка си с клиенти, като слага в тях някаква идентификация от кого са пристигнали. При излизане от Chat-а (с командата QUIT), сървърът получава и изпълнява команди за напускане на съответните клиенти и ги изтрива от списъците.

Ако всичко е протекло нормално, екранът на Alan в този момент трябва да изглежда по начин подобен на този:

Welcome to the UDP chat

SEND * Hi all!

Received message from 192.168.0.1: Hi all!

SEND 192.168.0.2 Hi Bob!

Received message from 192.168.0.2: Hi Alan, I am pleased to chat with you!

Received message from 192.168.0.2: Anybody for Counter Strike?

SEND 192.168.0.2 Bob, I don’t like this game. I have to work now.

SEND 192.168.0.2 Goodbye Bob.

SEND * Goodbye all.

Received message from 192.168.0.1: Goodbye all.

QUIT

Екранът на Bob в този момент трябва да изглежда по начин подобен на този:

Welcome to the UDP chat

Received message from 192.168.0.1: Hi all!

Received message from 192.168.0.1: Hi Bob!

SEND 192.168.0.1 Hi Alan, I am pleased to chat with you!

SEND * Anybody for Counter Strike?

Received message from 192.168.0.2: Anybody for Counter Strike?

Received message from 192.168.0.1: Bob, I don’t like this game. I have to work now.

Received message from 192.168.0.1: Goodbye Bob.

Received message from 192.168.0.1: Goodbye all.

SEND * Ok. I am leaving. The game session will start at 11.30 pm. Bye all.

Received message from 192.168.0.2: Ok. I am leaving. The game session will start at 11.30 pm. Bye all.

QUIT

Не е нужно да създавате графичен потребителски интерфейс. Достатъчно е да напишете конзолни приложения, които реализират описаната функционалност. Не е нужно да се грижите за синхронизация на конзолата (т.е. допустимо е докато пишете команда, съобщение, пристигнало от някъде, да се отпечата върху реда, в които пишете).

Забележка: В по-старите версии на настоящия документ проект 1.1 беше друг (Multicast

Chat), но се наложи да бъде променен заради необходимостта от работеща локална мрежа за да се използват Multicast сокети. Заради невъзможността на повечето от вас да разработват проекта си на компютър, който е включен в локална мрежа, се наложи условието му да бъде променено. Моля всички да работят по новото условие (UDP Chat Client/Server)!

Проект 1.2 – Simple Web Server:

Да се напише програма на Java, която използвайки възможностите на пакета java.net реализира прост Web-сървър, който обслужва TCP заявките на порт 80. Сървърът трябва да може да работи с няколко потребителя едновременно и да отговаря на следната команда:

GET [ресурс] HTTP/x.x – за извличане на файл от сървъра съгласно HTTP протокола

При заявка за достъп до директория на сървъра (например при поискване на /docs/), той трябва да връща динамично генериран HTML документ със списък от линкове към всички файлове и директории в съответната директория. При заявка за достъп до файл (например при поискване на /index.html), трябва да връща този файл. Ако файлът не съществува или не може да бъде отворен за четене, трябва да се връща грешка 404 съгласно HTTP протокола (виж примерите). Имената на ресурсите в GET-заявките могат да бъдат зададени както във вид на URL (например http://localhost/index.html), така и във вид на път и име до ресурса (например /welcome.gif). Ако името на един ресурс е зададено чрез URL, вашият сървър трябва да извлече от това URL само частта задаваща пътя и името на ресурса, като игнорира http:// частта и host-а. Вашият сървър трябва да приема за главна директория (Document root) някоя предварително фиксирана директория на от файловата система (например C:DOCUMENT_ROOT). Тази директория е физическото съответствие във файловата система на главната виртуална директория в сървъра. Тази главна виртуална директория означаваме с “/”. В нашия пример ако потребителят поиска от сървъра файла /docs/inetjava/InetJava.doc, това съответства на физическия файл C:DOCUMENT_ROOTdocsinetjavaInetJava.doc, а ако поиска директорията /, това съответства на физическата директория C:DOCUMENT_ROOT във файловата система на сървъра. На потребителя трябва да се дава достъп до всички файлове и поддиректории на Document root директорията и до никои други, т.е. разхождайки се по директориите, той трябва да не може да напусне главната виртуална директория.

При команда, която не е GET, да се върне отговор:

HTTP/1.0 501 Not Implemented

MIME-Version: 1.0

Content-type: text/html

Not implemented

Не е нужно да създавате графичен потребителски интерфейс. Достатъчно е да напишете конзолно приложение, което реализира описаната функционалност. Можете да тествате вашето приложение със стандартен Web-браузър (например Internet Explorer).

За улеснение може да се счита, че в имената на файловете и директориите няма интервали и специални символи (като “, ‘, :, /, и подобни).

Пример: Web-браузерът (клиентското приложение) се свързва на порт 80 към вашия Simple Web Server и задава няколко-редова текстова заявка. Например:

GET http://localhost/index.html HTTP/1.1

Accept: */*

Accept-Language: bg

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Host: localhost

Connection: Keep-Alive

Cache-Control: no-cache

Забележете, че последният ред на заявката е празен. Съгласно HTTP протокола този празен ред определя края на заявката. В края на първия ред е зададена версията на HTTP протокола която е използвана за формиране на заявката. Тя може да бъде както HTTP/1.1, така и HTTP/1.0. Независимо от тази версия вашият сървър трябва да връща отговорите си по протокол HTTP/1.0. Той трябва да отвори файла index.html, намиращ се в главната виртуална директория и да го върне на клиента в съответен отговор. Например:

HTTP/1.0 200 OK

MIME-Version: 1.0

Content-type: text/html

Welcome

Забележете че, между заглавната част на отговора и самия документ има оставен празен ред. Съгласно HTTP протокола този празен ред разделя хедъра от данните в отговора на HTTP GET заявка.

Вашият Web-сървър трябва да задава в хедъра на отговора в полето Content-type различни стойности в зависимост от разширението на поискания файл:

text/html – за *.htm, *.html файлове

image/gif – за *.gif файлове

image/jpg – за *.jpg файлове

application/zip – за *.zip файлове

text/plain – за всички останали типове файлове (*.*)

Освен това в този хедър на HTTP отговора трябва да се включи още един ред: “MIME-Version: 1.0”, който задава с коя версия на MIME стандарта да се тълкува реда “Content type: ...”.

Пример 2:

Заявка (изпратена от клиента за извеждане на файловете от директория /nakov):

GET /nakov HTTP/1.1

Accept: */*

Host: localhost

Отговор (върнат от сървъра):

HTTP/1.0 200 OK

MIME-Version: 1.0

Content-type: text/html

Directory listing of /nakov/

/nakov/InetJava.doc

/nakov/HTTP_protocol.zip

/nakov/JAVA_sockets.zip

Вижда се, че съгласно спецификацията на вашия Web-сървър, той връща списък с линкове към файлове в зададената директория (когато тя съществува, т.е. има физическо съответствие във файловата система). Забележете, че ресурсът поискан с командата GET /nakov HTTP/1.1 е зададен не като URL, а като път и име на ресурс. Тази команда е еквивалентна на командата GET http://localhost/nakov HTTP/1.1 в рамките на вашия Web-сървър.

Пример 3:

Заявка за несъществуващ файл (изпратена от клиента):

GET http://localhost/nakov/index.html HTTP/1.1

Accept: */*

Host: localhost

Отговор (върнат от сървъра):

HTTP/1.0 404 Not Found

MIME-Version: 1.0

Content-type: text/html

Requested document not found!

Пример 4:

Неподдържана заявка (изпратена от клиента):

POST http://localhost/nakov/index.php HTTP/1.0

Accept: */*

Отговор (върнат от сървъра):

HTTP/1.0 501 Not Implemented

MIME-Version: 1.0

Content-type: text/html

Not implemented

Проект 1.3 – Simple HTTP Proxy Server:

Да се напише програма на Java, която използвайки възможностите на пакета java.net реализира прост HTTP Proxy-сървър. Този Proxy-сървър не е нужно да е напълно работоспособен и трябва да реализира само частично действието на истинските Proxy-сървъри. Вашият сървър трябва да слуша на TCP порт 3128 и да обслужва HTTP GET заявки. Всяка HTTP GET заявка е от вида:

GET HTTP/x.y

където последният ред е празен (съгласно HTTP протокола). Например:

GET http://localhost/nakov/index.html HTTP/1.1

Accept: */*

Host: localhost

Proxy-сървърът, който трябва да напишете трябва да анализира зададената HTTP заявка, да извлича от нея Web-сървъра, за който тя е предназначена и да заменя версията на HTTP протокола от края на първия ред (HTTP/x.y) с HTTP/1.0. Получената нова HTTP заявка трябва да се изпраща за изпълнение към вече известния web-сървър (host). Полученият резултат трябва да се върне обратно на клиента. Web-сървърът трябва да се извлича от URL-то, ако присъства в него или от полето Host в хедъра на заявката в противен случай. Ако не е зададен порт, се подразбира порт 80. Всички HTTP GET заявки се преобразуват до HTTP/1.0 GET заявки, защото протоколът HTTP/1.0 е по-прост за използване. Схемата на действие на Proxy-сървъра е следната:

Клиентите изпращат HTTP GET заявки за достъп до отдалечени обекти не конкретно към Web-сървъра, на който те трябва да бъдат изпълнени, а към Proxy-сървъра, а той ги препраща и изпълнява на съответния Web-сървър и връща резултата на клиентите. (С понятията Web-сървър и HTTP сървър означаваме едно и също нещо – сървър, обслужващ заявки по протокол HTTP). След връщането на резултата на клиента, сокетът с него се затваря. За простота се предполага, че клиентът изпраща само HTTP GET заявки (т.е. такива, чийто първи ред започва с “GET ”). POST и HEAD заявки не се поддържат, което означава, че вашият Proxy-сървър не е напълно работоспособен, но това ограничение е поставено само за улеснение на задачата. Вашият Proxy-сървър трябва да може да обслужва няколко клиента едновременно. Не е нужно да създавате графичен потребителски интерфейс. Достатъчно е да напишете конзолно приложение, което реализира описаната функционалност.

В Интернет Proxy-сървъри се използват по няколко причини: за ускоряване достъпа до често използвани ресурси чрез кеширане; за ограничаване на достъпа до някои ресурси чрез филтриране; за достъп до отдалечени ресурси от мрежа, която е зад firewall.

Вашият Proxy-сървър трябва да спира достъпа до адреси в домейна “.bg”. При заявка за достъп до такъв адрес, да се връща грешка 403 – Forbidden. Вашият малък Proxy-сървър трябва да може да се използва от страндартен Web-браузър (например Internet Explorer). Той трябва да отпечатва на стандартния изход всички клиентски заявки и съответните им отговори. При заявка към сървър, който е недостъпен по някаква причина, трябва да се върне грешка 404 – Not Found, (въпреки, че това противоречи на стандарта, според който трябва да се върне грешка с друг код). Връщането на грешка става чрез отговор на заявка по HTTP протокола подобна на следната:

HTTP/1.0 404 Not Found

MIME-Version: 1.0

Content-type: text/html

Requested document not found!

където кодът на HTTP отговора е код на грешка (в случая 404).

Пример: Web-браузърът (клиентското приложение, в нашия случай Internet Explorer 6.0) се свързва на порт 3128 към вашия Simple Proxy Server и задава няколко-редова HTTP заявка. Например:

GET /index.html HTTP/1.1

Accept: */*

Accept-Language: bg

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Host: localhost

Connection: Keep-Alive

Cache-Control: no-cache

Вашият Proxy-сървър прочита клиентската заявка, променя версията на HTTP протокола (намира се на първия ред), извлича името на host-а, за който е предназначена тя, свързва се към този host (в нашия пример localhost) на порт 80 и подава преработената HTTP заявка на съответния Web-сървър. Забележете, че URL-то не съдържа host-а и той трябва да се вземе от реда “Host: localhost”. Ето и преработената заявка:

GET /index.html HTTP/1.0

Accept: */*

Accept-Language: bg

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Host: localhost

Connection: Keep-Alive

Cache-Control: no-cache

Web-сървърът, който слуша на порт 80 на локалната машина (localhost), връща в отговор на тази заявка поискания документ /index.html и затваря комуникационния сокет:

HTTP/1.1 200 OK

MIME-Version: 1.0

Content-type: text/html

Welcome to Nakov Web server

Този отговор, без изменения се изпраща на клиента и сокета за комуникация с него се затваря.

Пример 2: Web-браузърът се свързва на порт 3128 към вашия Proxy Server и задава следната HTTP заявка:

GET http://www.top.bg/ HTTP/1.0

Accept: */*

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Вашият Proxy-сървър прочита клиентската заявка, вижда че host-а (www.top.bg) е в домайна .bg и връща следния отказ от облужване:

HTTP/1.1 403 Forbidden

MIME-Version: 1.0

Content-type: text/html

Access to .BG domain is restricted.

Contact your system administrator for more details.

Този отговор, без изменения се изпраща на клиента. В случая host-а се взима от URL-то от първия ред на заявката.

Пример 3: Web-браузърът се свързва на порт 3128 към вашия Proxy Server и задава следната HTTP заявка:

GET / HTTP/1.1

Accept: */*

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Host: www.niama-takuv-server.com

Вашият Proxy-сървър прочита клиентската заявка и се опитва да се свърже към host-а “www.niama-takuv-server.com” на порт 80, но не успява, защото няма такъв сървър. Затова се на клиента се връща следната грешка:

HTTP/1.0 404 Not Found

MIME-Version: 1.0

Content-type: text/html

Requested document not found!

Хост-а се взима от полето Host на GET-заявката защото в URL-то на първия й ред го няма.

Пример 4: Web-браузърът (в нашия случай Opera 5.12) се свързва на порт 3128 към вашия Simple Proxy Server и задава следната HTTP заявка:

GET http://psstzing.dyndns.org:7001/ HTTP/1.1

User-Agent: Opera/5.12 (Windows 2000; U) [en]

Host: psstzing.dyndns.org:7001

Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */*

Accept-Language: en

Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0

Connection: Keep-Alive, TE

Вашият Proxy-сървър прочита клиентската заявка, променя версията на HTTP протокола, извлича името на host-а, за който е предназначена тя, свързва се към този host (в нашия случай host-а е psstzing.dyndns.org) на порт 7001 и подава преработената HTTP заявка:

GET http://psstzing.dyndns.org:7001/ HTTP/1.0

User-Agent: Opera/5.12 (Windows 2000; U) [en]

Host: psstzing.dyndns.org:7001

Accept: text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */*

Accept-Language: en

Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0

Connection: Keep-Alive, TE

Забележете, че вашият Proxy-сървър трябва да вземе името на host-а от реда “GET http://psstzing.dyndns.org:7001/ HTTP/1.1”, а не от полето “Host: psstzing.dyndns.org:7001”, защото URL-то на първия ред е пълно и съдържа host-а. Web-сървърът, който слуша на порт 7001 на адрес psstzing.dyndns.org връща в отговор на тази заявка поискания документ http://psstzing.dyndns.org:7001/ (документа по подразбиране) и затваря комуникационния сокет:

HTTP/1.1 200 OK

Date: Mon, 18 Mar 2002 09:27:51 GMT

Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod_ssl/2.7.1 OpenSSL/0.9.5a PHP/4.0.3pl1

Last-Modified: Mon, 20 Aug 2001 03:37:17 GMT

ETag: "c4fc-c2-3b8085ed"

Accept-Ranges: bytes

Content-Length: 194

Keep-Alive: timeout=15, max=100

Connection: Keep-Alive

Content-Type: text/html

Main page

Този отговор, без изменения се изпраща на клиента и сокета за комуникация с него се затваря. Забележете, че полето host от HTTP заявката може да съдържа IP адрес или host или host:port. Ако не е зададен порт, се подразбира 80 (стандартния порт за услугата HTTP).

Проект 1.4 – Simple Mail Server:

Да се напише програма на Java, която използвайки възможностите на пакета java.net реализира прост Mail-сървър. Програмата трябва да слуша на TCP портове 25 и 110 и да обслужва съответно SMTP и POP3 заявки съгласно SMTP и POP3 протоколите. Сървърът трябва да се състои от 2 части – SMTP сървър и POP3 сървър, които поддържат 3 фиксирани потребителя на системата съответно с имена user1, user2 и user3. За всеки от тях трябва да се създаде пощенска кутия (списък от текстови съобщения, който първоначално е празен). Тази пощенска кутия се използва за получаване и съхранение на съобщенията за този потребител. При осъществяване на връзка със SMTP сървъра, той трябва да поздравява клиентите си със съобщението: “220 Simple mail server”. След поздрава SMTP сървърът трябва да отговаря на следните команди:

HELO *hostname* – връща съобщение “250 Welcome *hostname*”.

MAIL FROM: – връща съобщение “250 *sender_email* Sender OK”.

RCPT TO:

– ако *recipient_email* е един от вградените потребители (user1, user2 или user3) връща съобщение “250 *recipient_email* Recipient OK“.

– ако *recipient_email* не е един от вградените потребители, връща “550 *recipient_email* Recipient not found on this server “.

DATA

– ако командата RCPT TO е завършила с успех (с код 250), връща съобщение “354 Enter mail, end with "." on a line by itself” след което прочита информацията, която клиента изпраща до срещане на първата точка сама на ред. След това връща съобщение “250 Message delivered” и добавя в пощенската кутия на потребителя-получател (взима името му от командадата RCPT TO) новото съобщение. Предполага се, че текстът на съобщението не може да съдържа редове, които започват със символа точка.

– ако командата RCPT TO е завършила с неуспех (с код, различен от 250), връща съобщение “503 Invalid state”.

QUIT – връща съобщение “221 Bye” и затваря сокета.

При непозната команда SMTP сървърът трябва да отговаря с: “500 Invalid command”. За командите MAIL FROM и RCPT TO не е задължително e-mail адресите да са заградени от символите “”. Трябва да се позволяват и двата варианта (със и без “”).

При осъществяване на връзка със POP3 сървъра, той трябва да поздравява клиентите си със съобщението: “+OK POP3 server ready”. След това POP3 сървърът трябва да отговаря на следните команди:

USER *username* – връща съобщение “+User OK, please send your password”.

PASS *password*

– ако username съвпада с *password* и съвпада с ”user1” или с “user2” или с “user3” връща съобщение “+OK”.

– в противен случай връща съобщение “-ERR Invalid username/password”.

LIST – връща информация за пощенската кутия на потребителя *username* в следния формат:

+OK X messages

1 YYY

2 YYY

...

X YYY

.

където X е броят съобщения в пощенската кутия на потребителя *username*, а на следващите X реда стоят числата от 1 до X следвани от размера в байтове YYY на всяко от съобщенията от 1 до X и на следващия ред има една точка (сигнализираща края на списъка).

RETR X

– ако X е валиден номер на съобщение, връща следния отговор:

+OK message X

– ако X не е валиден номер на съобщение, връща отговор “-ERR No such message”.

QUIT – връща съобщение “+OK Bye” и затваря сокета.

При невалидна команда трябва да се извежда съобщение “-ERR Invalid command”. За да бъдат изпълнявани командите LIST и RETR, е необходимо клиентът първо успешно да е преминал командите user и pass, т.е. да е получил отговор започващ с “+” и на двете. В противен случай командите LIST и RETR връщат съобщение за грешка “-ERR Invalid state”.

Вашият Mail-сървър трябва да бъде програма, която може да обслужва няколко

клиента едновременно както по SMTP, така и по POP3 протокола, като се грижи за синхронизация на достъпа до пощенските кутии (става въпрос за синхронизация на достъпа до пощенската кутия като структура от данни в паметта, а не за синхронизация на ниво сесия). Пощенските кутии трябва да стоят само в паметта на сървъра, без да се съхраняват на друго място (и разбира се, да губят съдържанието си при спиране на сървъра). Не е нужно да създавате графичен потребителски интерфейс. Достатъчно е да напишете конзолно приложение, което реализира описаната функционалност.

Пример: Отваряме връзка с Mail-сървъра на порт 25 (SMTP) и водим диалог със сървъра. Командите на клиента са дадени с префикс “ *Client:*”, а на сървъра с префикс “ *Server:*”.
В този пример потребителят се опитва да изпрати поща до непознат за сървъра адрес, а след това изпраща писмо до user1, писмо до user3 и още едно писмо до user1:

Server: 220 Simple mail server

Client: HELO www.nakov.com

Server: 250 Welcome www.nakov.com

Client: MAIL FROM:

Server: 250 mincho_penchov@yahoo.com Sender OK

Client: RCPT TO:

Server: 550 gincho_praschev@abv.bg Recipient not found on this server

Client: Skapan server!

Server: 500 Invalid command

Client: RCPT TO: user1

Server: 250 user1 Recipient OK

Client: DATA

Server: 354 Enter mail, end with "." on a line by itself

Client: Hi, user1!

Client: .

Server: 250 Message delivered

Client: DATA

Server: 503 Invalid state

Client: MAIL FROM: baba_yaga@the-hell.com

Server: 250 baba_yaga@the-hell.com Sender OK

Client: RCPT TO: user3

Server: 250 user3 Recipient OK

Client: DATA

Server: 354 Enter mail, end with "." on a line by itself

Client: Hi, user3!

Client: .

Server: 250 Message delivered

Client: MAIL FROM: baba_yaga@the-hell.com

Server: 250 baba_yaga@the-hell.com Sender OK

Client: RCPT TO: user1

Server: 250 user1Recipient OK

Client: DATA

Server: 354 Enter mail, end with "." on a line by itself

Client: Hi, user1,

Client: Greetings from Baba Yaga!

Client: .

Server: 250 Message delivered

Client: QUIT

Server: 221 Bye

След това отваряме връзка към Mail-сървъра, но този път на порт 110 (където се обслужва услугата POP3) и водим диалог със сървъра. Опитваме се да взезем с невалидна парола, влизаме успешно като потребител user1 и му прочитаме съобщенията. Отново командите на клиента са дадени с префикс “*Client:*”, а на сървъра с префикс “
Server:
”. Ето и диалога:

Server: +OK POP3 server ready

Client: USER pencho_ginchev

Server: +User OK, please send your password

Client: LIST

Server: -ERR Invalid state

Client: PASS tajna

Server: -ERR Invalid username/password

Client: USER user1

Server: +User OK, please send your password

Client: PASS user1

Server: +OK

Client: LIST

Server: +OK 2 messages

Server: 1 11

Server: 2 37

Server: .

Client: Samo tolkowa li bre? Iskam oshte e-mail!

Server: -ERR Invalid command

Client: RETR 1

Server: +OK message 1

Server: Hi, user1!

Server: .

Client: RETR 37

Server: -ERR No such message

Client: RETR 2

Server: +OK message 1

Server: Hi, user1,

Server: Greetings from Baba Yaga!

Server: .

Client: QUIT

Server: +OK Bye

Проект 1.5 – Simple File Client/Server:

Да се напише програма на Java, която използвайки възможностите на пакета java.net реализира прост файлов сървър. Да се напише програма-клиент за този сървър.

Сървърът трябва да дава възможност на потребителя да вижда файловете в текущата директория, да сменя текущата директория, да изтегля файлове и да изпраща файлове. Началната директория (Root directory) от която се стартира потребителската сесия

трябва да е фиксирана и сървърът трябва да дава достъп само до нея и нейните поддиректории. Файловият сървър трябва да слуша на TCP порт 2001 и да изпълнява следните команди, изпратени от клиента:

DIR, LIST, LS – извеждат всички файлове в текущата директория, по един на ред, като за всеки файл се дава името му и дължината му, а за всяка директория се дава името й и вместо дължина. Да се извежда още и информация за името на текущата директория, за броя на файловете и поддиректориите в нея и за общата дължина на показаните файлове. Форматът на изхода от командите DIR, LIST, LS (които са напълно еквивалентни) трябва да е оформен по следния образец:

+Listing directory /home/nakov

..

documents

inetjava

file1.txt 117

file3.zip 3225187

file2.doc 73812

Total 3 files (3299116 bytes), 2 subdirectories.

Ако няма файлове и директории, списъкът трябва да е празен, а на последния ред да пише “Total 0 files (0 bytes), 0 subdirectories”. Директорията е относителен път спрямо началната директория (Root directory). За улеснение може да се счита, че в имената на файловете и директориите няма интервали и специални символи (като “, ‘, :, /, и подобни). Специалната директория “..“ означава по-горната директория и се появява само ако има такава. Забележете, че изходът от командите DIR, LIST, LS завършва с точка сама на ред. Това ще ви е от помощ при реализацията на клиента.

CD *directory* – сменя текущата директория, като влиза в посочената. При успех връща съобщение “+Directory changed to *current_directory*”, а при невъзможност да се влезе в посочената директория, връща съобщение “-Invalid directory”.

GET *filename* – ако файлът с име *filename* от текущата директория не може да бъде отворен за четене (например ако не съществува), командата връща съобщение “-Invalid file”. Ако файлът се отвори успешно, се връща размера и съдържанието на файла по следния образец:

+File size and file contents follow:

filesize

Съдържанието на файла не е задължително да е текстово.

SEND *filename*

filesize

– изпраща файл с посоченото име *filename*, с посочената дължина *filesize* и с посоченото съдържание
на сървъра. Файлът се записва в текущата директория на сървъра. Изпратеното съдържание на файла трябва да е точно
filezise
байта. След получаване на файла, сървърът връща съобщение “+File
filename
stored”. Ако на сървъра има файл с това име или не е позволено писане в текущата директория (заради забрана от операционната система например) или поради друга причина файлът не може да се запише, сървърът трябва да върне съобщение за грешка “-Can not store file in current directory”.

HELP – връща кратка помощна информация завършваща със символа “.” сам на ред. Тази точка е индикатор за край на помощната информация. Примерен резултат от командата:

+Valid commands are:

DIR – Lists current directory

CD directory – Changes current directory

GET filename – Retrieves given file from current directory

SEND filename

filesize

– Sends given file to the server

HELP – Displays this help

QUIT – Closes this session

QUIT – връща съобщение “+Bye” и затваря връзката с клиента (затваря сокета).

Сървърът предполага, че клиентът е коректен и не изпраща неправилни команди (или параметри на команди). Сървърът трябва да позволява няколко клиента да се обслужват едновременно.

Клиентското приложение, което трябва да напишете трябва да се свързва с файловия сървър и да препраща командите, които вие пишете, към него, а върнатия резултат от тези команди да отпечатва на конзолата. За улеснение на потребителя при получаване на файлове е необходимо при успешен отговор на сървъра на командата GET *filename*, върнатият файл да се запише в работната директория на клиента. За улеснение на изпращането на файлове, клиентът трябва да разбира командата

UPLOAD *filename*

която трябва автоматично да преобразува в команда към сървъра SEND с подходящи параметри. Ако файлът *filename* не съществува в работната директория на клиента, командата UPLOAD трябва да се игнорира, а на потребителя да бъде изведено съобщение “Can not find file *filename*”. Работната директория на клиента може да фиксирана (например C:TEMP) или да се използа текущата (тази от която се стартира Java виртуалната машина).

Както за сървъра, така и за клиента, не е нужен графичен интерфейс. Достатъчно е да се напише конзолно приложение, което покрива описаната функционалност.

Пример: Клиентското приложение се свързва със файловия сървър и потребителят разглежда файловете по сървъра, след това издърпва няколко файла и накрая изпраща един файл на сървъра. Междувременно той не знае командите и ги разучава по метода с опити и грешки:

Client: ALO

Server: -Invalid command

Client: USER pencho

Server: -Invalid command

Client: HELP

Server: +Valid commands are:

Server: DIR – Lists current directory

Server: CD directory – Changes current directory

Server: GET filename – Retrieves given file from current directory

Server: SEND filename

Server: filesize

Server: – Sends given file to the server

Server: HELP – Displays this help

Server: QUIT – Closes this session

Server: .

Client: OK, now I know the commands. Lets have some fun...

Server: -Invalid command

Client: DIR

Server: +Listing directory /

Server:

Server: docs

Server: books

Server: InetJava

Server: PrAnKa

Server:

Server: Total 0 files (0 bytes), 4 subdirectories.

Server: .

Client: CD docs

Server: +Directory changed to /docs

Client: DIR

Server: +Listing directory /docs

Server:

Server: ..

Server: passwords.txt 55

Server: HTTP_protokol.zip 211458

Server:

Server: Total 2 files (211503 bytes), 0 subdirectories.

Server: .

Client: GET passwords.txt

Server: +File size and file contents follow:

Server: 55

Server: user: niki

Server: password: niki

Server:

Server: user: test

Server: password: test123

Client: GET SMTP-POP3-protokols.zip

Server: -Can not find file SMTP-POP3-protokols.zip

Client: CD books

Server: -Invalid directory

Client: CD /books

Server: -Invalid directory

Client: CD /

Server: -Invalid directory

Client: CD ..

Server: +Directory changed to /

Client: CD ..

Server: -Invalid directory

Client: CD books

Server: +Directory changed to /books

Client: DIR

Server: +Listing directory /books

Server:

Server: ..

Server: more_books

Server: Thinking_in_Java.zip 2812397

Server: SQL_in_21_days.zip 332865

Server: readme.txt 131

Server:

Server: Total 3 files (3145393 bytes), 1 subdirectories.

Server: .

Client: GET readme.txt

Server: +File size and file contents follow:

Server: 131

Server: Thinking_in_Java.zip – Thinking in Java (in PDF), Bruce Eckel

Server: SQL_in_21_days.zip – Teach Yourself SQL in 21 Days, Second Edition

Client: SEND test.txt

Client: 5

Client: test *(символите са 5, защото има символ n за край на ред след думата “*test*”)*

Server: +File test.txt stored

Client: LIST

Server: +Listing directory /books

Server:

Server: ..

Server: more_books

Server: test.txt 5

Server: Thinking_in_Java.zip 2812397

Server: SQL_in_21_days.zip 332865

Server: readme.txt 131

Server:

Server: Total 4 files (3145398 bytes), 1 subdirectories.

Server: .

Client: QUIT

Server: +Bye

Последна версия на този документ можете да намерите от сайта на курса:

http://inetjava.sourceforge.net .

Коментари по задачите можете да публикувате във форума на курса:

http://sourceforge.net/forum/forum.php?forum_id=152811

Автор на проектите: Светлин Наков

Последна промяна: 27.03.2002