# Министерство науки и высшего образования Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования

## «Новосибирский государственный технический университет»

NSTU_Logo_blue

## Кафедра теоретической и прикладной математики

### Лабораторная работа №2 по дисциплине « Разработка web-приложений и распределенных информационных систем»

|  |  |  |
| --- | --- | --- |
| эмблема_светлая.png |  |  |
| Факультет: | ПМИ |
| Группа: ПМИ-62 | |
| Студенты: | Мамонова Е.В., Ершов П.К., Цыденов З. Б. |
| Преподаватель: | Цыгулин А. А. |
|  | | |

Новосибирск

2019

1. **Задание**

Реализовать клиентское приложение, осуществляющее пересылку между всеми подключенными клиентами пакетов определённой структуры, указанной преподавателем.

Должна выполнятся пересылка хеша файла к пользователю, хеш которого ближе всего к хешу файла.

Ближайшим пользователем считает пользователь, чей хэш даёт наименьшее число с хэшем файла после операции XOR.

1. **Анализ задачи**

**Алгоритм программы**

Объявить сокет.

Связать сокет с портом.

Запросить имя пользователя.

Запросить имя файла.

Считать содержимое файла.

Получить хэши от имени и содержимого файла.

Поместить полученные хэши в массив.

Рассчитать id пользователя по его IP адресу.

Прослушивать порт.

Если пришло сообщение

Распаковать сообщение.

Если пришёл запрос на проверку активности:

Собрать сообщение для ответа (id + имя).

Отослать сообщение.

Иначе, если пришёл ответ на проверку об активности:

Вывести id и имя.

Внести id и ip в массивы для обработки.

Если пришёл запрос на сохранение ссылки на файл:

Записать информацию о файле в файл с ссылками (хэш от файла, имя пользователя).

По мере получения id от соседей.

Искать ближайшего соседа для выбранного хэша от файла.

Собирать сообщение с выбранным хэшем от файла.

Отсылать по ближайшему адресу.

1. **Структура данных приложения**

Вся программа хранится в файле web\_lab2.js

В одной папке с рабочим файлом программы хранятся файл с информационной карточкой и файл, куда помещаются хэши от соседей.

Для запуска программы необходимо ввести с консоли: **node web\_lab2.js**

Программа потребует ввести имя пользователя и имя файла.

Дальнейшие действия будут происходит автоматически.

1. **Текст программы**

**web\_lab2.js**

var dgram = require('dgram'); //Модуль dgram обеспечивает реализацию сокетов UDP датаграмм

const fs = require("fs");//модуль для работы с файлом

var dgramSocket = dgram.createSocket("udp4"); //Для UDP-сокетов заставляет dgram.Socket прослушивать сообщения дейтаграммы на именованном порту

var crypto = require('crypto'); // crypto - модуль шифрования. Подключаю его для создания хеша

var readline = require('readline');// Предоставляет интерфейс для чтения данных из читаемого потока по одной строке за раз

var msg\_size = 1; // Количество байт под тип сообщения

var msg\_length\_size = 2; // Количество байт под длину сообщения

var interval = 2000; // Каждые 2 секунды запрос

var port = 1024; // Номер порта

var readline = readline.createInterface(process.stdin, process.stdout, null); // Для считывания текста с консоли

var onlineUser = []; // Массив со всеми пользователями онлайн

var allHesh = []; // Массив со всеми хэшами от файла

var alladdr = []; // Массив со всеми адресами

var allId = []; // Массив с id (хэшами) от соседей

var n\_hesh = [];//Массив с хэшами от файлов от соседей

var userName; // имя пользователя

var userId;// хэш пользователя

var flag = false; // Флаг дял онлайн\оффлайн пользователя

var ip\_addr = '255.255.255.255'; // Адрес, с которого отправляется запрос

var hesh\_file = '';//хэш от содержимого файла

var home\_ip = '';//ip пользователя

var request = 0; // Запрос

var response = 1; // Ответ

var mess = 2; // Сообщение с хэшем

// Определяем ip адрес компьютера. К Первым 3-м цифрам ip добавляем 255 (широковещательный)

require('dns').lookup(require('os').hostname(), function (err, add, fam)

{

for (var i = 0; i < add.length; i++)

home\_ip += add[i];

})

// Сокет начал прослушивание

dgramSocket.on('listening', function ()

{

var ServerAddress = dgramSocket.address();

console.log('Listening ' + ServerAddress.address + ":" + ServerAddress.port);

});

// Сокет создан

dgramSocket.bind(port, function ()

{

dgramSocket.setBroadcast(true);

console.log('Broadcast on.');

readline.question('Your name: ', (name) => {

readline.question('Name of file: ', (info) => {

// Заношу информацию об имени и закрываю входной поток

userName = name;

readline.close();

let data = fs.readFileSync(info + ".txt", "utf8");

var lineReader = require('readline').createInterface({

input: require('fs').createReadStream(info + ".txt")

});

var line = '';

//var inform = '';

lineReader.on('line', function (line) {

console.log('Line from file:', line);

//inform += line;

});

// Заношу информацию об имени пользователя

user\_name = crypto.createHash('md5').update(info).digest(); // name (присваиваю хеш md5)

//Заношу информацию о пользователе

hesh\_file = crypto.createHash('md5').update(line).digest(); // line (присваиваю хеш md5)

//Помещаю хэши в массив для работы с ними.

allHesh.push(user\_name);

allHesh.push(hesh\_file);

// Заношу информацию об id пользователя

userId = crypto.createHash('md5').update(home\_ip).digest(); // id (присваиваю хеш md5)

loopFunction(); // Предварительно вызываем, чтобы не ждать интервального времени перед первым вызовом

setInterval(loopFunction, interval); // Задаём периодичность выполнения для функции

});

});

});

// ищет минимум в массиве и возвращает его индекс

function arrayMin(arr)

{

var len = arr.length, min = Infinity;

var cr;

while (len--)

{

if (arr[len] < min)

{

min = arr[len];

cr = len;

}

}

return cr;

};

//провит операцию XOR с битами хэша

function metric(hash1, hash2)

{

let ret = Buffer.from(hash1)

for (let i =0;i<hash1.length;i++)

ret[i]^=hash2[i]

return ret

}

//Проверяет есть ли элемент в массиве возвращая 1 - да и 0 - нет

function find(arr, elem)

{

var len = arr.length;

var fl = 0;

while (len--)

{

var bf = arr[len];

if ( roughScale(bf.toString("hex"), 16).toString() == roughScale(elem.toString("hex"), 16).toString())

{

fl = 1;

break;

}

}

return fl;

};

//переводит хэш в 10-е значение (для поиска минимума)

function roughScale(x, base)

{

var parsed = parseInt(x, base);

if (isNaN(parsed)) { return 0 }

return parsed;

}

// Сокет получил сообщение

dgramSocket.on('message', function (msg, rinfo) {

//Если не введено имя

if (userName === undefined)

return

var msgType = msg[0]; // Определяем тип сообщения

// Для запроса. Формируем дейтаграмму

if (msgType === request)

{

var rmsg = Buffer.allocUnsafe(msg\_size + msg\_length\_size + Buffer.byteLength(userName) + userId.length); // Собираем сообщение, чтоб ответить

rmsg[0] = response;

rmsg.fill(userId, msg\_size, msg\_size + userId.length)

rmsg.writeUInt16BE(Buffer.byteLength(userName),msg\_size + userId.length );

rmsg.fill(userName, msg\_size + userId.length + msg\_length\_size ); // + name

dgramSocket.send(rmsg, 0, rmsg.length, rinfo.port, rinfo.address);

}

//Для расшифровки

if (msgType === response)

{

var msgLength; //длина сообщения

msgLength = msg.readUInt16BE(msg\_size + userId.length);

var msgText = msg.toString('utf-8', msg\_size + userId.length + msg\_length\_size); // Извлекаем текст сообщения

var id = Buffer.from(msg.slice(msg\_size, msg\_size + userId.length));

var msg\_addr = rinfo.address;

var msg\_port = rinfo.port;

var flag = 0;

// заносим ip в массив (кроме домашнего, для проверки работоспособности убрать второе условие)

if(!find(alladdr, msg\_addr) && !find(alladdr, home\_ip))

alladdr.push(msg\_addr);

// заносим id в массив (кроме домашнего, для проверки работоспособности убрать второе условие)

if(!find(allId, id) && !find(alladdr,home\_ip))

allId.push(id);

onlineUser.push(msgText + " " + id.toString('hex')); // Добавляем онлайн пользователя в массив

}

//Если в сообщении есть хэш от файла

if (mess === msgType)

{

var msgLength; //длина сообщения

msgLength = msg.readUInt16BE(msg\_size + userId.length);

var msgText = msg.toString('utf-8', msg\_size + userId.length + msg\_length\_size); // Извлекаем текст сообщения

var id = Buffer.from(msg.slice(msg\_size, msg\_size + userId.length));

var msg\_addr = rinfo.address;

var flag = 0;

n\_hesh.push(msgText + " " + id.toString('hex') + " " + msg\_addr.toString()); // Добавляем онлайн пользователя в массив

}

});

function loopFunction()

{

// Если не подключились, пропускаем этап вывода на экран информации об online пользователях

// Подключение в данном случае - первый вызов этой функции

if (flag === true)

{

// Выводим информацию об online пользователях, которую сформировали с предыдущей рассылки

console.log("Online:");

// Очищаем массив с информацией от соседей

while (onlineUser.length)

console.log(onlineUser.pop());

console.log("\n");

console.log("Hesh:");

// Очищаем массив с хэшами от файлов от соседей

while (n\_hesh.length)

{

fs.appendFileSync("data.txt", "||" + n\_hesh[n\_hesh.length - 1] + "|| \n", "UTF-8");

console.log(n\_hesh.pop());

}

}

else

{

console.log("Connection...");

flag = true;

}

console.log("Request...");

checkOnline(); // Запрос на проверку онлайн-пользователей

//весь бред ниже нужен для поиска адреса, на который нужно отослать хэш от файла

//сначала берём один из хэшей от файла.

//затем, с помощю массива с хэшами от соседей (id) и хэша от файла создаём новый массив через опреацию XOR

//полученный массив переводим в 10 систему

//в 10-м массиве ищем индес минимального элемента - это индекс в массиве со всеми ip соседей

//заносим найденный индекс в массив-карту, который нужен для отправки хэшей по адресам

//в массиве-карте номер элемента массива это номер хэша от файла, а число в элементе массива это индекс в массиве ip

var i = 0;

var send\_map = []; // Массив со всеми номерами элементов allHesh, alladdr, allport для сборки сообщений с хэшами

send\_map.length = allHesh.length;

while( i < allHesh.length)

{

var buffID = [];

var j = 0;

while(j < allId.length)

{

buffID[j] = metric(allId[j], allHesh[i]);

j++;

}

var find\_buff = [];

find\_buff.length = allId.length;

j = 0;

while(j < allId.length)

{

find\_buff[j] = roughScale(buffID[j].toString("hex"), 16).toString();

j++;

}

send\_map[i] = arrayMin(find\_buff);

i++;

}

for(var k = 0; k < send\_map.length; k++)

send\_hesh(alladdr[send\_map[k]], k); // хэш под номером k отправляем по адресу под номеров send\_map[k]

console.log("\n");

}

function send\_hesh(ip, index)

{

var rmsg = Buffer.allocUnsafe(msg\_size + msg\_length\_size + Buffer.byteLength(userName) + allHesh[index].length); // Собираем сообщение, чтоб ппослать хэш

rmsg[0] = mess;

rmsg.fill(allHesh[index], msg\_size, msg\_size + allHesh[index].length)

rmsg.writeUInt16BE(Buffer.byteLength(allHesh[index]),msg\_size + allHesh[index].length);

rmsg.fill(userName, msg\_size + userId.length + msg\_length\_size ); // + name

dgramSocket.send(rmsg, 0, rmsg.length, port, ip);

}

// Функция широковещательной рассылки

function checkOnline()

{

var requestMsg = Buffer.allocUnsafe(msg\_size + msg\_length\_size + Buffer.byteLength(userName) + userId.length); // Формируем сообщение для проверки пользователей

requestMsg[0] = request;

requestMsg.fill(userId, msg\_size, msg\_size + userId.length)

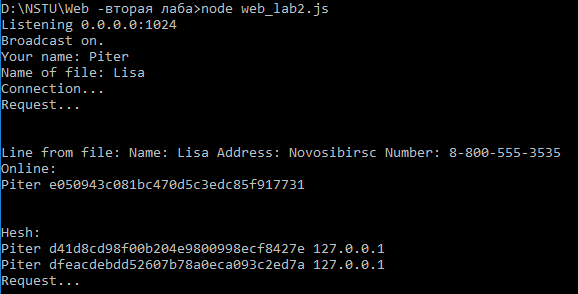
requestMsg.writeUInt16BE(Buffer.byteLength(userName),msg\_size + userId.length );

requestMsg.fill(userName, msg\_size + userId.length + msg\_length\_size ); // + name

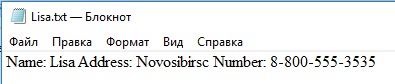
dgramSocket.send(requestMsg, 0, requestMsg.length, port, ip\_addr);// Отсылаем запрос

}

1. **Пример работы программы**



Lisa.txt



Data.txt

