Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

02.03.02 Фундаментальная информатика и

информационные технологии

**ОТЧЕТ**

по учебной практике

по направлению 02.03.02 «Фундаментальная информатика и информационные технологии», направленность (профиль) – «Системное программное обеспечение», квалификация – бакалавр, программа прикладного бакалавриата,

форма обучения – очная, год начала подготовки (по учебному плану) – 2018

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. ИС-842  «11» июля 2020 г. | \_\_\_\_\_\_\_\_\_\_ | Есикова А.Д |
| Оценка «отлично» |  |  |
| Руководитель практики от университета  д.т.н., профессор Кафедры ВС  «11» июля 2020 г. | \_\_\_\_\_\_\_\_\_\_ | Павский К.В. |

Новосибирск 2020

**ПЛАН-ГРАФИК ПРОВЕДЕНИЯ ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ**

Тип практики: учебная практика по получению первичных профессиональных умений и навыков, в том числе получение умений и навыков научно-исследовательской деятельности

Способ проведения практики: стационарная

Форма проведения практики: дискретно по периодам проведения практики

Тема: Разработка терминального клиент-серверного приложения с использованием библиотеки pthreads.

Содержание практики

|  |  |
| --- | --- |
| Наименование видов деятельности | Дата  (начало – окончание) |
| Общее ознакомление со структурным подразделением СибГУТИ, вводный инструктаж по технике безопасности | 03.02.20-15.02.20 |
| Выдача задания на практику, определение конкретной индивидуальной темы, формирование плана работ | 27.02.20-7.03.20 |
| Работа с библиотечными фондами Кафедры вычислительных систем, сбор и анализ материалов по теме практики | 09.03.20-28.03.20 |
| Выполнение работ в соответствии с составленным планом  - реализация программного кода для работы сервера  - реализация программного кода для работы клиента  - написание заголовочного файла приложения  - оформление файла, содержащего основные функции клиент-серверного приложения | 06.04.20-06.06.20 |
| Анализ полученных результатов и произведенной работы, составление отчета по практике | 06.07.20-11.07.20 |

Согласовано:

|  |  |  |
| --- | --- | --- |
| Руководитель практики от университета  д.т.н., профессор Кафедры ВС | \_\_\_\_\_\_\_\_\_\_ | Павский К.В |

**ЗАДАНИЕ НА ПРАКТИКУ**

Требуется реализовать сетевое терминальное клиент-серверное приложение на базе протокола TCP/IP с использованием библиотеки pthreads.

**ВВЕДЕНИЕ**

Сеть Интернета позволяет общаться со всеми, кто подключён к ней. На сегодняшний день у большинства есть такая возможность, а способы выходить в Интернет становятся разнообразнее. Понимание основных принципов работы сети позволят лучше понимать внутрисетевые процессы, выявлять причину плохих соединений и исправлять их и разрабатывать приложения для удобного общения между пользователями в рамках одного или нескольких подключений.

**ВЗАИМОДЕЙСТВИЕ УСТРОЙСТВ В РАМКАХ ОДНОЙ СЕТИ**

Помимо главной сети Интернета, какие-либо устройства могут быть подключены между собой через домашнюю микросеть. Каждое устройство в рамках такой сети использует какие-нибудь ресурсы либо само предоставляет их. Существуют соединение, работающие только в одну сторону: клиент подключается к серверу и принимает данные.

Чтобы подключится к устройству и передать ему какие-либо данные, нужно знать не только его адрес и чтобы то устройство знало адрес данного, но и сокет. Сокет позволяет не только отправлять данные, но и принимать их. [2]

С помощью сокета, или же по-другому гнезда, процесс на одной машине может взаимодействовать с процессом на другой. Это возможно благодаря тому, что когда к гнезду обращаются по IP-адресу хост-машины и номеру порта, задаётся адрес, который в рамках этой сети является уникальным. [1]

Рассмотрим принцип передачи данных. Клиент устройства создаёт сокет, ищет заданный адрес в сети, соединяется с устройством, передаёт данные и отключается. Эта последовательность действий едина для каждого клиента. [2]

Каждое гнездо, с которым было установлено соединение, действует по схеме клиент/сервер.

Сокеты без соединения назначается адрес, чтобы процесс использовал их и мог посылать сообщения других процессам.[1]

**ОСНОВНЫЕ ИНТЕРФЕЙСЫ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ**

Вернёмся к описанию установления соединения клиента с другими. Для начало нужно создать гнездо определённого типа для указанного домена и протокола. Этот шаг осуществляется с помощью Socket. Bind задаст имя, а listen позволит задать очередь для клиентов. Чтобы принять запрос на соединение от клиентского сокета, используют accept, в то время как send позволит передать данные, а recv – принять их. Для закрытия гнезда, то есть прекращения соединения, используют shutdown или close. [1]

В конечном итоге получится схема, приведённая ниже.

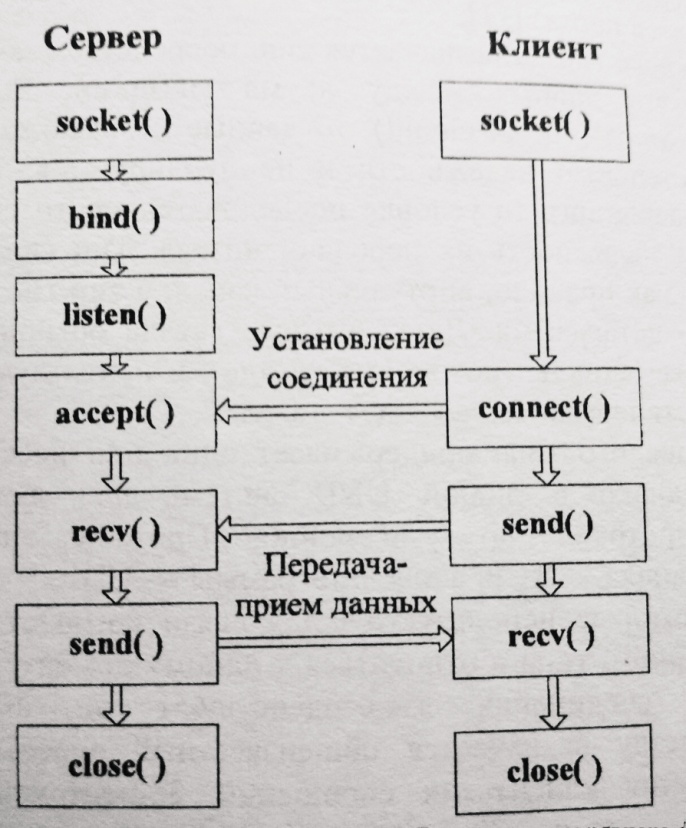


Рис 1. Схема взаимодействия клиента и сервера

**ОСНОВНЫЕ ФУНКЦИИ ДЛЯ СОЗДАНИЯ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ**

Функция socket(). Организует канал связи с другим устройством через заданным правилом именования гнезда и формат адреса протокола. Эта функция образует единый интерфейс между семи протоколами в Linux/UNIX. [2]

Функиция bind(). Присваивает гнезду имя. Подключение очередей к сетевым потокам происходит только при выполнении системного вызова bind().

Функция listen(). Используется для создания очереди. Вызывается серверным процессом для создания гнезда, определяемого для установления соединения. [1]

Функция connect(). Производит установку с серверным гнездом. Важно отметить, что потоковое гнездо может соединяться с клиентом только один раз.

Функция accept(). Вызывается в серверном процессе для установления соединения с клиентским гнездом и возвращает дескриптор нового гнезда, с чьим помощью серверный процесс взаимодействует только с данным клиентом.

Функция recv(). Принимает сообщения через заданное гнездо.

Функция close(). Закрывает соединение между серверным и клиентским гнёздами. [1]

В рамках данной практической работы будет применяться непосредственно локальный сетевой адрес 127.0.0.1.

**ПРИМЕНЕНИЕ ПОТОКОВ ДЛЯ ОБЕСПЕЧЕНИЯ ПАРАЛЛЕЛЬНОЙ РАБОТЫ СЕРВЕРА.**

**НЕКОТОРЫЕ ФУНКЦИИ БИБЛОТЕКИ PTHREAD**

Использование потоков, предоставляемых устройством сервера, позволит принимать клиентов без дополнительных временных затрат. Так, каждому клиенту будет предоставлен свой поток, в котором клиент будет взаимодействовать с сервером.

Операционная система Linux предусматривает несколько механизмов синхронизация. Рассмотрим только один, который понадобиться для создания приложения, – Мьютекс.

Мьютекс в потоках используется для взаимоисключающего доступа к разделяемому элементу данных. В рамках данной работы будут использоваться такие функции, как pthread\_mutex\_lock и pthread\_mutex\_unlock. Когда используется первая функция, первый поток, вызвавший её, будет работать с данными дальше, в то время, как другие не смогут её вызвать. Unlock разблокирует первый ожидающий своей очереди поток. [1]

Теперь рассмотрим другие основные функции, которые будут использоваться при дальнейшей разработке приложений.

Функция pthread\_create(). Создаёт потоки. После создания, в нём начинают выполняться заданные функции.

**РЕЗУЛЬТАТЫ РАБОТЫ**

****

Рис 2. Демонстрация работы приложения

**ЗАКЛЮЧЕНИЕ**

В результате учебной практики разработано терминальное клиент-серверное приложение «Чат», работающее по локальной сети. Были изучены основные принципы работы протокола TCP/IP и его инструменты.

В дальнейшем, данное приложение может быть использовано для общения сотрудников между собой в рамках рабочей сети компании.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Павский К.В. Протоколы TCP/IP и разработка сетевых приложений. – Новосибирск: СибГУТИ, 2013. – 130 с.
2. Создание сетевых приложений в среде Linux. : Пер. с англ.— М. : Издательский дом"Вильяме", 2001. — 464 с.: ил. — Парал: тит. англ.

**ПРИЛОЖЕНИЕ**

* + - 1. Листинг программы

/\*

\* Clieserv.h.

\*

\*/

#ifndef CLIESERV\_H

#define CLIESERV\_H

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <pthread.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <string.h>

//#include <signal.h>

#define MAX\_CLIENT 5

#define Buffer\_size 2048

#define NAME\_LEN 15

typedef struct

{

struct sockaddr\_in addres;

int sockfd;

int uid;

char name [NAME\_LEN];

} clients;

int flag = 0;

int sockfd = 0;

char name[32];

static int cliCount = 0;

static int uid = 10;

pthread\_mutex\_t clientsMutex = PTHREAD\_MUTEX\_INITIALIZER;

int Socket(int domain, int type, int protocol);

void Bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

void Listen(int sockfd, int backlog);

int Accept(int sickfd, struct sockaddr \*addr, socklen\_t \*addrlen);

void Connect(int socket, const struct sockaddr \*addr, socklen\_t addrlen);

void Inet\_pton(int af, const char \*src, void \*dst);

int Sendto(int sockfd, const void \*buf, size\_t len, int flags, struct sockaddr \*addr, socklen\_t addrlen);

int Resvfrom(int sockfd, void \*buf, size\_t len, int flags, struct sockaddr \*addr, socklen\_t \*addrlen);

void printIpAddr(struct sockaddr\_in addr);

void QueueRemove(int uid);

void AddInQueue(clients \*cl);

void sendMessage(char \*text, int uid);

void \*handleClient();

int Pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine)(void \*), void \*arq);

void RecvMsg();

void SendMsg();

void catch\_ctrl\_c\_and\_exit(int sig);

void str\_trim\_lf (char\* arr, int length);

void str\_overwrite\_stdout();

#endif

/\*

\* Clieserv.c.

\*

\*/

#ifndef CLIESERV\_C

#define CLIESERV\_C

#include "Clieserv.h"

clients \*Clients[MAX\_CLIENT];

int Socket(int domain, int type, int protocol)

{

int res = socket(domain, type, protocol);

if (res == -1)

{

perror("socket failed\n");

exit(1);

}

return res;

}

void Bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen)

{

int res = bind(sockfd, addr, addrlen);

if (res == -1)

{

perror("bind failed\n");

exit(1);

}

}

void Listen(int sockfd, int backlog)

{

int res = listen(sockfd, backlog);

if (res == -1)

{

perror("listen failed");

exit(1);

}

}

int Accept(int sickfd, struct sockaddr \*addr, socklen\_t \*addrlen)

{

int res = accept(sickfd, addr, addrlen);

if (res == -1)

{

perror("accept failed\n");

exit(1);

}

return res;

}

void Connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen)

{

int res = connect(sockfd, addr, addrlen);

if (res == -1)

{

perror("connect failed\n");

exit(1);

}

}

void Inet\_pton(int af, const char \*src, void \*dst)

{

int res = inet\_pton(af, src, dst);

if (res == 0)

{

printf("inet\_pton failed: src does not contain a character strling representing a valid network addres in specified addres family\n");

exit(1);

}

if (res == -1)

{

perror("inet\_pton failed\n");

exit(1);

}

}

int Write(int handle, void \*buf, int count)

{

int res = write (handle, buf, count);

if(res < count || res < 0)

{

perror("write failed\n");

exit(1);

}

return res;

}

int Sendto(int sockfd, const void \*buf, size\_t len, int flags, struct sockaddr \*addr, socklen\_t addrlen)

{

int res = sendto(sockfd, buf, len, flags, addr, addrlen);

if (res < 0)

perror("Sendto failed\n");

return res;

}

int Recv(int sockfd, void \*buf, size\_t len, int flags)

{

int res = recv(sockfd, buf, len, flags);

if (res < 0)

perror("resvfrom failed\n");

return res;

}

void printIpAddr(struct sockaddr\_in addr)

{

printf("%d.%d.%d.%d",

addr.sin\_addr.s\_addr & 0xff,

(addr.sin\_addr.s\_addr & 0xff00) >> 8,

(addr.sin\_addr.s\_addr & 0xff0000) >> 16,

(addr.sin\_addr.s\_addr & 0xff000000) >> 24);

}

void str\_trim\_lf (char\* arr, int length)

{

for (int i = 0; i < length; i++)

if (arr[i] == '\n')

{

arr[i] = '\0';

break;

}

}

void AddInQueue(clients \*cl)

{

pthread\_mutex\_lock(&clientsMutex);

for (int i = 0; i < MAX\_CLIENT; i++)

if (!Clients[i])

{

Clients[i] = cl;

break;

}

pthread\_mutex\_unlock(&clientsMutex);

}

void QueueRemove(int uid)

{

pthread\_mutex\_lock(&clientsMutex);

for (int i = 0; i < MAX\_CLIENT; i++)

if (Clients[i])

if (Clients[i]->uid == uid)

{

Clients[i] = NULL;

break;

}

pthread\_mutex\_unlock(&clientsMutex);

}

void sendMessage(char \*text, int uid)

{

pthread\_mutex\_lock(&clientsMutex);

for (int i = 0; i < MAX\_CLIENT; i++)

if (Clients[i])

if (Clients[i]->uid != uid)

if(write(Clients[i]->sockfd, text, strlen(text)) < 0)

{

perror("ERROR: write to descriptor failed");

break;

}

pthread\_mutex\_unlock(&clientsMutex);

}

void \*handleClient(void \*arg)

{

char buf[Buffer\_size];

char name[NAME\_LEN];

int leaveFlags;

cliCount++;

clients \*cli = (clients \*)arg;

if(recv(cli->sockfd, name, NAME\_LEN, 0) <= 0 || strlen(name) < 2 || strlen(name) >= NAME\_LEN-1)

{

printf("Didn't enter the name.\n");

leaveFlags = 1;

}

else

{

strcpy(cli->name, name);

sprintf(buf, "%s has joined\n", cli->name);

printf("%s", buf);

sendMessage(buf, cli->uid);

}

bzero(buf, Buffer\_size);

while(1)

{

if (leaveFlags)

break;

Recv(cli->sockfd, buf, Buffer\_size, 0);

if (strlen(buf) > 0)

{

sendMessage(buf, cli->uid);

str\_trim\_lf(buf, strlen(buf));

printf("%s -> %s", buf, cli->name);

}

else if (strcmp(buf, "exit") == 0)

{

sprintf(buf, "%s has left\n", cli->name);

printf("%s", buf);

sendMessage(buf, cli->uid);

leaveFlags = 1;

}

else

{

printf("-1\n");

leaveFlags = 1;

}

bzero(buf, Buffer\_size);

}

close(cli->sockfd);

QueueRemove(cli->uid);

free(cli);

cliCount--;

pthread\_detach(pthread\_self());

return 0;

}

int Pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine)(void \*), void \*arq)

{

int res = pthread\_create(thread, attr, start\_routine, arq);

if (res != 0)

{

perror("pthread\_create failed\n");

exit(1);

}

return res;

}

void str\_overwrite\_stdout()

{

printf("%s", "> ");

fflush(stdout);

}

void catch\_ctrl\_c\_and\_exit(int sig)

{

flag = 1;

}

void SendMsg()

{

char message[Buffer\_size] = {};

char buffer[Buffer\_size + NAME\_LEN] = {};

while(1)

{

str\_overwrite\_stdout();

fgets(message, Buffer\_size, stdin);

str\_trim\_lf(message, Buffer\_size);

if (strcmp(message, "exit") == 0)

break;

else

{

sprintf(buffer, "%s: %s\n", name, message);

send(sockfd, buffer, strlen(buffer), 0);

}

bzero(message, Buffer\_size + NAME\_LEN);

bzero(buffer, Buffer\_size);

}

catch\_ctrl\_c\_and\_exit(2);

}

void RecvMsg()

{

char message[Buffer\_size] = {0};

while (1)

{

int receive = recv(sockfd, message, Buffer\_size, 0);

if (receive > 0)

{

printf("%s", message);

str\_overwrite\_stdout();

}

else if (receive == 0)

break;

memset(message, 0, sizeof(message));

}

}

#endif

/\*

\* Server.c.

\*

\*/

#include "Clieserv.h"

#include "Clieserv.c"

pthread\_mutex\_t clients\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

int main(int argc, char \*argv[])

{

if (argc != 2)

{

perror("ERROR, no port provided\n");

exit(1);

}

char \*ip = "127.0.0.1";

int port = atoi(argv[1]);

int connfd = 0;

//listenfd = serverAdr

struct sockaddr\_in serverAdr = {0};

struct sockaddr\_in clientAddr = {0};

pthread\_t tid;

int server = Socket(AF\_INET, SOCK\_STREAM, 0);

serverAdr.sin\_family = AF\_INET;

serverAdr.sin\_addr.s\_addr = inet\_addr(ip);

serverAdr.sin\_port = htons(port);

Bind(server, (struct sockaddr \*) &serverAdr, sizeof serverAdr);

Listen(server, 5);

printf("\n\*\*\*\*\*\*\*\*\*\*Welcome\*\*\*\*\*\*\*\*\*\*\n");

while (1)

{

socklen\_t clieLen = sizeof(clientAddr);

connfd = Accept(server, (struct sockaddr \*) &clientAddr, &clieLen);

if((cliCount +1) == MAX\_CLIENT)

{

printf("Maximum client connected. Connection Rejected\n");

printIpAddr(clientAddr);

printf(":%d\n", clientAddr.sin\_port);

close(connfd);

continue;

}

clients \*cli = (clients \*)malloc(sizeof(clients));

cli->addres = clientAddr;

cli->sockfd = connfd;

cli->uid = uid++;

//Add client to queue

AddInQueue(cli);

Pthread\_create(&tid, NULL, &handleClient, (void \*)cli);

sleep(1);

}

return 0;

}

/\*

\* Client.c.

\*

\*/

#include "Clieserv.h"

#include "Clieserv.c"

int main(int argc, char \*argv[])

{

if (argc != 2)

{

perror("ERROR, no port provided\n");

exit(1);

}

char \*ip = "127.0.0.1";

int port = atoi(argv[1]);

//signal(SIGINT, catch\_ctrl\_c\_and\_exit);

printf("Please enter your name: ");

fgets(name, 32, stdin);

str\_trim\_lf(name, strlen(name));

if (strlen(name) > 32 || strlen(name) < 2)

{

printf("Name must be less than 30 and more than 2 characters.\n");

exit(1);

}

struct sockaddr\_in server = {0};

sockfd = Socket(AF\_INET, SOCK\_STREAM, 0); //протокол

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = inet\_addr(ip);

server.sin\_port = htons(port);

// Connect to Server

Connect(sockfd, (struct sockaddr \*)&server, sizeof server);

// Send name

send(sockfd, name, NAME\_LEN, 0);

printf("\n\*\*\*\*\*\*\*\*\*\*Welcome\*\*\*\*\*\*\*\*\*\*\n");

pthread\_t send\_msg\_thread;

Pthread\_create(&send\_msg\_thread, NULL, (void \*) SendMsg, NULL);

pthread\_t recv\_msg\_thread;

Pthread\_create(&recv\_msg\_thread, NULL, (void \*) RecvMsg, NULL);

while (1)

if(flag)

{

printf("\nBye\n");

break;

}

close(sockfd);

return 0;

}