Министерство науки и образования РФ  
Новосибирский государственный технический университет  
Кафедра ТПИ

Лабораторная работа №8

по дисциплине «Управление ресурсами в вычислительных системах»

Факультет: ПМИ

Группа: ПМИ-31

Выполнили: Махнёва Е. А.

Соловьёв Б. С.

Преподаватели: Хайленко Е. А.  
 Филиппова Е. В.

Вариант: 9

1. **Цель работы**

Практическое освоение механизма сокетов. Построение TCP-соединений

для межпроцессного взаимодействия программ Клиента и Сервера в модели

“клиент-сервер”.

1. **Задание**

Организовать взаимодействие типа клиент-сервер. Клиент делает запрос

серверу на выполнение какой-либо команды. Сервер выполняет эту команду и

возвращает результаты клиенту.

1. **Описание использованных системных вызовов**

**int socket(int domain, int type, int protocol) <sys/socket.h>**

Создаёт новый сокет типа type в домене domain, основанный на протоколе protocol. Возвращает файловый дескриптор нового сокета или -1 при ошибке.

**int connect(int fd, CONST\_SOCKADDR\_ARG addr, socklen\_t len) <sys/socket.h>**

Открывает соединение на сокете fd к узлу addr (длина адреса которого len байт). Для сокетов без установления соединения (вроде UDP) устанавливает адрес по умолчанию для отправки и отбрасывает входящие пакеты не с этого адреса. Возвращает 0 при успехе, либо -1 при ошибке.

**int bind(int fd, CONST\_SOCKADDR\_ARG addr, socklen\_t len) <sys/socket.h>**

Назначает сокету fd локальный адрес addr (длина адреса которого len байт).

**int listen(int fd, int n) <sys/socket.h>**

Запускает режим прослушивания на сокете fd. N одновременных запросов на соединение будут ставиться в очередь, дальнейшие будут отклоняться.

Возвращает 0 при успехе, либо -1 при ошибке.

**int accept(int fd, SOCKADDR\_ARG addr, socklen\_t \*\_\_restrict addr\_len) <sys/socket.h>**

Ожидает входящих соединений на сокете fd. Когда происходит соединение, для него создаётся новый сокет, в addr записывается адрес подключившегося клиента (длина адреса которого addr\_len байт). Возвращает дескриптор нового сокета, либо -1 при ошибке.

**ssize\_t read(int fd, void \*buf, size\_t n) <unistd.h>**

Выполняет чтение из файла по дескриптору fd не более n байт в память buf. Возвращает количество фактически прочитанных байт. Тип ssize\_t эквивалентен short.

**ssize\_t write(int fd, const void \*buf, size\_t n) <unistd.h>**

Записывает в файл по дескриптору fd не более n байт из памяти buf. Возвращает количество фактически записанных байт.

**int close(int fd) <unistd.h>**

Закрывает дескриптор файла. Фактически это может быть как файл, так и канал.

1. **Текст программы**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdexcept> // for exceptions

#include <fcntl.h>

#define BUFSIZE 512

#define SOCKET\_QUEUE 10

#define FORK\_ERROR 1 // Exit status for failed fork()

#define NOFLAGS 0

#define SYSCALL\_FAILURE -1

#define SV\_PORT 3030 // it's 666 in sixth basis

#define inet\_atoa(in\_addr\_t) inet\_ntop(AF\_INET,(void\*)&in\_addr\_t,buf+BUFSIZE-INET\_ADDRSTRLEN-1,INET\_ADDRSTRLEN)

bool server\_mode = false;

FILE \*logfile = stderr;

char buf[BUFSIZE];

typedef unsigned char byte; // this is missing in standard libs for whatever reason.

typedef struct { byte flags; byte code; } wait\_t; // union for wait result

// process creation parameters

typedef struct {

char \*command; // executable file to be found in PATH

char \*argv0; // argv[0] string, required for exec-() functions

char \*arg1, \*arg2; // first arg, second arg. no more due to complexity

int in, out, err; // descriptors of standard streams

} param\_t;

// Connect socket to remote address using its IP and port.

int connect\_to(int sockfd, in\_addr\_t ip\_addr, ushort port) {

sockaddr\_in s\_addr; // структура с информацией о сервере

s\_addr.sin\_family = AF\_INET;

s\_addr.sin\_port = htons(port);

s\_addr.sin\_addr.s\_addr = ip\_addr;

return connect(sockfd, (sockaddr\*)&s\_addr, sizeof(s\_addr));

}

// Bind socket locally to become a server.

int bind\_to(int sockfd, ushort port, in\_addr\_t addr = INADDR\_ANY) {

sockaddr\_in s\_addr; // структура с информацией о сервере

s\_addr.sin\_family = AF\_INET;

s\_addr.sin\_addr.s\_addr = addr; // 0.0.0.0 by default

s\_addr.sin\_port = htons(port);

return bind(sockfd, (sockaddr\*)&s\_addr, sizeof(s\_addr));

}

// Read single row from server list and return ready-to-use IP.

bool get\_server(char \*str, in\_addr\_t &ip) {

char addr[BUFSIZE];

in\_addr ip\_addr;

if (sscanf(str, "%s", addr) < 1)

return false;

if (!inet\_aton(addr, &ip\_addr)) // функция перевода ip-адреса в нужный тип

return false;

ip = ip\_addr.s\_addr; // take only IP value

return true;

}

// Fill param\_t with process creation data.

void create\_startup\_param(

param\_t \*param,

char \*command,

char \*arg1,

char \*arg2 = NULL,

int out = STDOUT\_FILENO,

int err = STDERR\_FILENO,

int in = STDIN\_FILENO) {

param->command = command;

param->argv0 = command;

param->arg1 = arg1;

param->arg2 = arg2;

param->in = in;

param->out = out;

param->err = err;

}

// Overlays current executing process with COMMAND.

int shell\_exec(param\_t \*param) {

dup2(param->out, STDOUT\_FILENO); // this is identical to fcntl(param->out, F\_DUPFD, 1)

dup2(param->err, STDERR\_FILENO);

dup2(param->in, STDIN\_FILENO);

int ret = execlp(param->command, param->argv0, param->arg1, param->arg2, NULL);

return ((ret == SYSCALL\_FAILURE) ? -1 : 0); // return -1 on failure, 0 otherwise

}

// Create new thread in new process via fork() and run callback in it.

pid\_t fork\_thread(int(func(param\_t \*param)), void \*param = NULL) {

int childpid;

switch (childpid = fork()) {

case SYSCALL\_FAILURE:

fprintf(stderr, "[shell]\t Could not fork to subcommand '%s'\n", ((param\_t\*)param)->command);

exit(FORK\_ERROR);

break;

case 0:

exit(func((param\_t\*)param)); // run callback and exit

break;

default:

fprintf(stderr, "[shell]\t Process %d created for '%s'\n", childpid, ((param\_t\*)param)->command);

}

return childpid;

}

// Wait till ALL running subcommands end

int join\_threads() {

pid\_t wait\_pid;

wait\_t wait\_ret;

int exec\_result = 0;

while ((wait\_pid = wait(&wait\_ret)) > 0) {

printf("[shell]\t Process %d exited with code %d\n", wait\_pid, wait\_ret.code);

exec\_result |= wait\_ret.code; // OR all codes

}

return exec\_result;

}

int perform\_client(char \*server\_ip) {

/\*

1. create socket

2. connect to server

\*/

int sockfd;

in\_addr\_t ip;

if (!get\_server(server\_ip, ip)) {

printf("[x] Invalid server IP\n");

return EXIT\_FAILURE;

}

try {

if ((sockfd = socket(PF\_INET, SOCK\_STREAM, SOL\_IP)) == -1) // получаем дескриптор сокета

throw std::runtime\_error("Could not create socket");

if (connect\_to(sockfd, ip, SV\_PORT) < 0) { // соединяемся с сервером

sprintf(buf, "Could not connect to server '%s:%d'", inet\_atoa(ip), SV\_PORT);

throw std::runtime\_error(buf);

}

printf("Enter command: ");

fgets(buf, BUFSIZE - 1, stdin); // формируем сообщение

buf[strlen(buf) - 1] = '\0';

if (write(sockfd, buf, BUFSIZE - 1) == SYSCALL\_FAILURE) // отправляем сообщение серверу

throw std::runtime\_error("Server unexpectedly dead");

//read(sockfd, &msg, sizeof(msg)); // получаем сообщение сервера; blocking mode

ssize\_t cnt, cnt2;// = recv(sockfd, &buf, BUFSIZE - 1, MSG\_PEEK);

//dup2(sockfd, STDOUT\_FILENO);

do {

//cnt = sendfile(sockfd, STDOUT\_FILENO, NULL, BUFSIZE);

cnt = recv(sockfd, buf, BUFSIZE - 1, NOFLAGS);

cnt2 = write(STDOUT\_FILENO, buf, cnt);

} while (cnt > 0 && cnt2 > 0);

}

catch (std::runtime\_error e) {

fprintf(logfile, "[!] %s\n", e.what());

}

close(sockfd); // finally

return EXIT\_SUCCESS;

}

int perform\_server() {

/\*

1. create socket

2. bind locally

3. listen

4. after each accept do actions

\*/

int sockfd;

try {

if ((sockfd = socket(PF\_INET, SOCK\_STREAM, SOL\_IP)) < 0) // получение дескриптора сокета сервера

throw std::runtime\_error("Cannot create socket");

if (bind\_to(sockfd, SV\_PORT) < 0) // связываем сокет с ip-адресом и портом

throw std::runtime\_error("Error in bind");

if (listen(sockfd, SOCKET\_QUEUE) < 0) // открываем доступ для клиентов

throw std::runtime\_error("Could not set listen mode");

fprintf(logfile, "[i] Server started on port %d\n", SV\_PORT);

}

catch (std::runtime\_error e) {

fprintf(logfile, "[x] Server: %s\n", e.what());

close(sockfd);

return EXIT\_FAILURE;

}

// ждем подключений клиентов, устанавливаем соединение

int cl\_sockfd;

while ((cl\_sockfd = accept(sockfd, NULL, NULL)) > 0) { // **TODO: move work to thread**

// **TODO: add loop**

read(cl\_sockfd, buf, BUFSIZE - 1); // получаем сообщение клиента

if (strncmp(buf, "exit", 4) == 0) {

close(cl\_sockfd);

fprintf(logfile, "[i] Server was stopped by user command\n");

break;

}

param\_t param;

create\_startup\_param(&param, buf, NULL, NULL, cl\_sockfd); // WARNING: duplicate dup2()

fork\_thread(shell\_exec, &param);

join\_threads();

close(cl\_sockfd); // закрываем дескриптор на сокет клиента

}

close(sockfd); // закрываем дескриптор на сокет сервера

return EXIT\_SUCCESS;

}

int main(int argc, char \*\*argv) {

char \*host = NULL;

int e;

while ((e = getopt(argc, argv, ":c:d")) != SYSCALL\_FAILURE) {

//fprintf(logfile, "Got argument: %c = %s\n", e, optarg);

switch (e) {

case 'c': host = optarg; break;

case 'd': server\_mode = true; break;

case '?': fprintf(logfile, "[!] Unknown command-line argument with value: %s\n", optarg); exit(1); break;

}

};

if (server\_mode)

return perform\_server();

if(host != NULL)

return perform\_client(host);

return printf("Usage:\n-c <hostname>\t - connect to <hostname>\n"

"-d\t\t - run as server, default port is %d\n", SV\_PORT);

}

1. **Тестирование**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Команда | Клиент | Сервер |
| 1 | ls | [striker@centos7x64 Debug]$ ./lab8\_rm\_solobor -c127.0.0.1  Enter command: ls  a.txt  b.txt  lab5\_rm\_solobor  lab6\_rm\_solobor  lab7\_rm\_solobor  lab8\_rm\_solobor  main5.dep  main5.o  main6.dep  main6.o  main7.dep  main7.o  main8.dep  main8.o | [shell] Process 5855 created for 'ls'  [shell] Process 5855 exited with code 0 |
| 2 | pwd | [striker@centos7x64 Debug]$ ./lab8\_rm\_solobor -c127.0.0.1  Enter command: pwd  /tmp/vs2010/labs\_rm\_solobor/Debug | [shell] Process 5858 created for 'pwd'  [shell] Process 5858 exited with code 0 |
| 3 | exit | [striker@centos7x64 Debug]$ ./lab8\_rm\_solobor -c127.0.0.1  Enter command: exit | [i] Server was stopped by user command |