**26. Server**

Реалізувати клієнт-серверну систему, що дозволяє клієнтам посилати повідомлення на сервер (потік байтів та можливість посилати широкі символи wchar). Повідомлення також містять ідентифікатори інших клієнтів, і сервер якщо бачить підєднавшегося відповідного клієнта пересилає йому ці повідомлення. Проект передбачає текстовий опис роботи програми (документація) та реферат по створенню клієнт/серверу на Сі/Сі++ .

**Файли в директорії Server**

client.c

server.c

helper.c

helper.h

**client.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <string.h>

#include <errno.h>

#include <pthread.h>

#include "helper.h"

#define MAXLINE 1024 */\* максимальна довжина \*/*

char prompt[]="Chatroom> ";

int flag=0;

*/\**

*команди*

*\*/*

void usage(){

printf("-h довідка\n");

printf("-a IP адрес сервера[Required]\n");

printf("-p номер порта сервера[Required]\n");

printf("-u введіть ваше імʼя[Required]\n");

}

*/\**

*\* @brief-: підключає клієнт до сервера*

*\* NOTE-: функція THE проходить список, щоб найди сокет*

*\* @port-: номер порта*

*\* @hostname-: ip адрес сервера*

*\* @return -: дескриптор файла підключення*

*\*/*

int connection(char\* hostname, char\* port){

int clientfd,rc;

struct addrinfo hints, \*listp, \*p;

memset(&hints, 0, sizeof(struct addrinfo));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM; */\* Тільки підключення \*/*

hints.ai\_flags |=AI\_ADDRCONFIG;

hints.ai\_flags |= AI\_NUMERICSERV; *//використання фіксованого порта*

if ((rc = getaddrinfo(hostname, port, &hints, &listp)) != 0) {

fprintf(stderr,"невірний хост або порт\n");

return -1;

}

for (p = listp; p; p = p->ai\_next) {

*/\* Створення дескриптора сокета \*/*

clientfd = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol);

if (clientfd < 0) {

continue; */\* Наступний сокет \*/*

}

*/\* Підключення до сервера \*/*

if (connect(clientfd, p->ai\_addr, p->ai\_addrlen) != -1) {

break; */\* Success \*/*

}

*/\* Не вдалось підкдючитися, спробуйте інше \*/*

if (close(clientfd) < 0) {

fprintf(stderr, "open\_clientfd: close failed: %s\n",

strerror(errno));

return -1;

}

}

*/\* Очистка \*/*

freeaddrinfo(listp);

if (!p) { */\* Всі підключення провалені \*/*

return -1;

}

else { */\* Останнє підключення пройшло успішно \*/*

return clientfd;

}

}

*// відповідь сервера*

void reader(void\* var){

char buf[MAXLINE];

rio\_t rio;

int status;

int connID=(int)var;

*// ініціалізація структури даних rio*

rio\_readinitb(&rio, connID);

while(1){

while((status=rio\_readlineb(&rio,buf,MAXLINE)) >0){

*// помилка*

if(status == -1)

exit(1);

if(!strcmp(buf,"\r\n")){

break;

}

*// вихід з сервера*

if (!strcmp(buf,"exit")){

close(connID);

exit(0);

}

if (!strcmp(buf,"start\n")){

printf("\n");

}

else

printf("%s",buf);

}

*// Підказка чату*

printf("%s",prompt);

fflush(stdout);

}

}

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

char \*address=NULL,\*port=NULL,\*username=NULL;

char cmd[MAXLINE];

char c;

pthread\_t tid;

*// Розбір аргументів командної лінії*

while((c = getopt(argc, argv, "hu:a:p:u:")) != EOF){

switch(c){

*// довідка*

case 'h':

usage();

exit(1);

break;

*// адрес сервера*

case 'a':

address=optarg;

break;

*// порт сервера*

case 'p':

port=optarg;

break;

*// імʼя*

case 'u':

username=optarg;

break;

default:

usage();

exit(1);

}

}

if(optind == 1 || port == NULL || address == NULL || username == NULL){

printf("Недопустиме\n");

usage();

exit(1); }

int connID=connection(address,port);

if(connID == -1){

printf("Неможливо підʼєднатися до сервера\n");

exit(1);

}

*// добавити нову строку*

sprintf(username,"%s\n",username);

*// відправити серверу ваше імʼя*

if(rio\_writen(connID,username,strlen(username)) == -1){

perror("Неможливо відправити інформацію");

close(connID);

exit(1);

}

*// потік для читання відповіді сервера*

pthread\_create(&tid,NULL,reader, (void\*)connID);

*// підказка чату*

printf("%s",prompt);

while(1){

*// читання команди*

if ((fgets(cmd, MAXLINE, stdin) == NULL) && ferror(stdin)) {

perror("fgets error");

close(connID);

exit(1);

}

*// відправлення запиту серверу*

if (rio\_writen(connID,cmd,strlen(cmd)) == -1){

perror("not able to send the data");

close(connID);

exit(1);

}

}

}

**server.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <string.h>

#include <errno.h>

#include <pthread.h>

#include "helper.h"

#define bufsize 1000

*// блокування до глобального доступа до даних*

pthread\_mutex\_t mutex;

struct client{

char \*name;

int confd;

struct client \*next;

};

struct client \*header=NULL;

*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*/\**

*\* @brief-: додавання користувача до глобального DATA\_STRUCTURES*

*\* INSERTION AN HEAD -> O(1) complexity*

*\*/*

void add\_user(struct client \*user){

if(header == NULL){

header=user;

user->next=NULL;

}

else{

user->next=header;

header=user;

}

}

*/\**

*\* @brief-: видалення клієнта з глобального списку*

*\* O(n) complexity*

*\*/*

void delete\_user(confd){

struct client \*user=header;

struct client \*previous=NULL;

*// ідентифікація користувача*

while(user->confd!=confd){

previous=user;

user=user->next;

}

if(previous == NULL)

header=user->next;

else

previous->next=user->next;

free(user->name);

free(user);

}

*/\**

*\* @brief-: призначає сокет списку на заданий номер порту*

*\* NOTE-: функція проходить по списку, щоб знайти відповідне підключення до сокета для сервера*

*\* @port-: номер порта*

*\* @return -: дескриптор файлу списку*

*\*/*

int connection(char \* port){

struct addrinfo \*p, \*listp, hints;

int rc,listenfd,optval=1;

*// ініціалізувати на нуль*

memset(&hints,0,sizeof(struct addrinfo));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM; */\* Тільки підключення \*/*

hints.ai\_flags =AI\_ADDRCONFIG|AI\_PASSIVE;

hints.ai\_flags |= AI\_NUMERICSERV; *//використання фіксованого порта*

if ((rc = getaddrinfo(NULL, port, &hints, &listp)) != 0) {

fprintf(stderr,"get\_address номер порта %s невірний\n",port);

return -1;

}

*// проходження списку доступних підключень*

for (p = listp; p; p = p->ai\_next) {

listenfd = socket(p->ai\_family, p->ai\_socktype, p->ai\_protocol);

if (listenfd < 0) {

continue; */\* Збій сокета, спробуйте наступний \*/*

}

*/\* Усуває помилку "Адреса вже використовується" з прив'язування \*/*

setsockopt(listenfd, SOL\_SOCKET, SO\_REUSEADDR, (const void \*)&optval,sizeof(int));

*//прив’язка сокета, повертає 0 у випадку успіху*

if (bind(listenfd, p->ai\_addr, p->ai\_addrlen) == 0) {

break; */\* Успіх \*/*

}

if (close(listenfd) < 0) { */\* Не вдалося прив’язати, спробуйте наступне \*/*

fprintf(stderr, "open\_listenfd збій закриття: %s\n",

strerror(errno));

return -1;

}

}

*// уникнення витоку пам'яті*

freeaddrinfo(listp);

if (!p) { */\* Збій всіх підключень \*/*

return -1;

}

*// встановлення відставання на 1024 , бажане значення*

*// налащтування сокета на listen*

if (listen(listenfd, 1024) < 0) {

close(listenfd);

return -1;

}

return listenfd;

}

*/\**

*\* відправлення повідомлення всім користувачам*

*\*/*

void send\_msg(int confd,char\* msg, char\* receiver, char\* sender){

char response[bufsize];

struct client \*user=header;

if(receiver == NULL)

while (user != NULL){

if (user->confd == confd){

strcpy(response,"повідомлення відправлено\n\r\n");

rio\_writen(user->confd,response,strlen(response));

}

else{

sprintf(response,"старт\n%s:%s\n\r\n",sender,msg);

rio\_writen(user->confd,response,strlen(response));

}

user=user->next;

}

else{

while (user != NULL){

if(!strcmp(user->name,receiver)){

sprintf(response,"старт\n%s:%s\n\r\n",sender,msg);

rio\_writen(user->confd,response,strlen(response));

strcpy(response,"повідомлення відправлено\n\r\n");

rio\_writen(confd,response,strlen(response));

return;

}

user=user->next;

}

strcpy(response,"користувача не знайдено\n\r\n");

rio\_writen(confd,response,strlen(response));

}

}

void evaluate(char \*buf ,int confd ,char \*username){

char response[bufsize];

char msg[bufsize];

char receiver[bufsize];

char keyword[bufsize];

*// очистка буфера*

msg[0]='\0';

receiver[0]='\0';

keyword[0]='\0';

struct client \*user=header;

if(!strcmp(buf,"help")){

sprintf(response,"msg \"text\" : відправити повідомлення всім користувачам онлайн\n");

sprintf(response,"%smsg \"text\" user : відправити повідомлення певному користувачу\n",response);

sprintf(response,"%sonline : отримати ім’я всіх користувачів онлайн\n",response);

sprintf(response,"%squit : вийти з чату\n\r\n",response);

rio\_writen(confd,response,strlen(response));

return;

}

*// отримати ім’я користувача*

if (!strcmp(buf,"online")){

*// очистка буфера*

response[0]='\0';

*// глобальний доступ має бути виключним*

pthread\_mutex\_lock(&mutex);

while(user!=NULL){

sprintf(response,"%s%s\n",response,user->name);

user=user->next;

}

sprintf(response,"%s\r\n",response);

*// глобальний доступ має бути ексклюзивним*

pthread\_mutex\_unlock(&mutex);

rio\_writen(confd,response,strlen(response));

return;

}

if (!strcmp(buf,"вихід")){

pthread\_mutex\_lock(&mutex);

delete\_user(confd);

pthread\_mutex\_unlock(&mutex);

strcpy(response,"exit");

rio\_writen(confd,response,strlen(response));

close(confd);

return;

}

sscanf(buf,"%s \" %[^\"] \"%s",keyword,msg,receiver);

if (!strcmp(keyword,"повідомлення")){

pthread\_mutex\_lock(&mutex);

send\_msg(confd,msg,receiver,username);

pthread\_mutex\_unlock(&mutex);

}

else {

strcpy(response,"Неправильна команда\n\r\n");

rio\_writen(confd,response,strlen(response));

}

}

*/\**

*\* @brief-: функція обробляє вхідних клієнтів одночасно*

*\* @vargp-: покажчик на дескриптор файлу підключення*

*\*/*

void\* client\_handler(void \*vargp ){

char username[bufsize];

rio\_t rio;

struct client \*user;

long byte\_size;

char buf[bufsize];

*// відʼєднання потока від пірів*

*// тому більше не потрібно*

*// закінчувати основним потоком*

pthread\_detach(pthread\_self());

*// збереження з'єднання fd у стеку функцій*

int confd = \*((int \*)vargp);

rio\_readinitb(&rio, confd);

*// зчитувати імʼя користувача як один рядок, -1 для обробки помилок*

if( (byte\_size=rio\_readlineb(&rio,username,bufsize)) == -1){

close(confd);

free(vargp);

return NULL;

}

*// видалити новий рядок*

username[byte\_size-1]='\0';

*// призначити простір у глобальній структурі*

user=malloc(sizeof(struct client));

*// error handling*

if (user == NULL){

perror("пам'ять не може бути призначена");

close(confd);

free(vargp);

return NULL;

}

*// user->name=username небезаечний*

*// оскільки локальний стек може бути доступний одноранговим потокам*

user->name=malloc(sizeof(username));

memcpy(user->name,username,strlen(username)+1);

user->confd=confd;

*// заблокувати*

pthread\_mutex\_lock(&mutex);

add\_user(user);

*// розблокувати*

pthread\_mutex\_unlock(&mutex);

*// прочитати відповідь клієнта*

while((byte\_size=rio\_readlineb(&rio,buf,bufsize)) >0){

*// видалити новий рядок*

buf[byte\_size-1]='\0';

*// вжити відповідних заходів*

evaluate(buf,confd,username);

}

return NULL;

}

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

struct sockaddr\_storage clientaddr;

socklen\_t clientlen;

int listen=-1;

char host[1000];

char \*port="80";

int \*confd;

pthread\_t tid;

if (argc > 1)

port=argv[1];

*// створення дескриптора файлу підключення*

listen= connection(port);

*// Збій підключення*

if(listen == -1){

printf("connection failed\n");

exit(1);

}

printf("waiting at localhost and port '%s' \n",port);

*// цикл, щоб продовжувати приймати клієнтів*

while(1){

confd=malloc(sizeof(int));

\*confd=accept(listen, (struct sockaddr \*)&clientaddr, &clientlen);

printf("A new client is online\n");

*// призначити окремий потік для роботи з новим клієнтом*

pthread\_create(&tid,NULL,client\_handler, confd);

}

}

**helper.c**

// цей файл реалізує допоміжну функцію, необхідну клієнтському та серверному коду

#include<stdlib.h>

#include<stdio.h>

#include"helper.h"

#include <unistd.h>

#include <errno.h>

#include <string.h>

ssize\_t rio\_readn(int fd, void \*usrbuf, size\_t n) {

size\_t nleft = n;

ssize\_t nread;

char \*bufp = usrbuf;

while (nleft > 0) {

if ((nread = read(fd, bufp, nleft)) < 0) {

if (errno != EINTR) {

return -1; /\* errno встановлюється read() \*/

}

/\* Перерваний поверненням обробника sig, знову виклик read() \*/

nread = 0;

} else if (nread == 0) {

break; /\* EOF \*/

}

nleft -= (size\_t)nread;

bufp += nread;

}

return (ssize\_t)(n - nleft); /\* Return >= 0 \*/

}

/\*

\* rio\_writen - Надійний запис n байтів (небуферизовані)

\*/

ssize\_t rio\_writen(int fd, const void \*usrbuf, size\_t n) {

size\_t nleft = n;

ssize\_t nwritten;

const char \*bufp = usrbuf;

while (nleft > 0) {

if ((nwritten = write(fd, bufp, nleft)) <= 0) {

if (errno != EINTR) {

return -1; /\* errno встановлюється write() \*/

}

/\* Переривається поверненням обробника sig, виклик write() знову \*/

nwritten = 0;

}

nleft -= (size\_t)nwritten;

bufp += nwritten;

}

return (ssize\_t)n;

}

/\*

\* rio\_read - Це оболонка для функції Unix read(), яка.

\* передає min(n, rio\_cnt) байт із внутрішнього буфера в

\* буфер користувача, де n - кількість байт, запрошених користувачем, а

\* rio\_cnt - кількість непрочитаних байт у внутрішньому буфері.

\* При вході, rio\_read() поповняє внутрішній буфер через виклик

\* функції read(), якщо внутрішній буфер пустий.

\*/

static ssize\_t rio\_read(rio\_t \*rp, char \*usrbuf, size\_t n) {

size\_t cnt;

while (rp->rio\_cnt <= 0) { /\* Заповнити якщо buffer пустий \*/

rp->rio\_cnt = read(rp->rio\_fd, rp->rio\_buf, sizeof(rp->rio\_buf));

if (rp->rio\_cnt < 0) {

if (errno != EINTR) {

return -1; /\* errno встановлюється read() \*/

}

/\* Перерваний обробником сигналу, нічого не робити \*/

} else if (rp->rio\_cnt == 0) {

return 0; /\* EOF \*/

} else {

rp->rio\_bufptr = rp->rio\_buf; /\* Перзавантажити buffer ptr \*/

}

}

/\* Копіювання min(n, rp->rio\_cnt) байтів з внутрішнього buf в користувацький buf \*/

cnt = n;

if ((size\_t)rp->rio\_cnt < n) {

cnt = (size\_t)rp->rio\_cnt;

}

memcpy(usrbuf, rp->rio\_bufptr, cnt);

rp->rio\_bufptr += cnt;

rp->rio\_cnt -= cnt;

return (ssize\_t)cnt;

}

/\*

\* rio\_readinitb - Зʼєднання дескриптора з буфером зчитування і скидання

\*/

void rio\_readinitb(rio\_t \*rp, int fd) {

rp->rio\_fd = fd;

rp->rio\_cnt = 0;

rp->rio\_bufptr = rp->rio\_buf;

}

/\*

\* rio\_readnb - Надійно зчитує n байтів (buffered)

\*/

ssize\_t rio\_readnb(rio\_t \*rp, void \*usrbuf, size\_t n) {

size\_t nleft = n;

ssize\_t nread;

char \*bufp = usrbuf;

while (nleft > 0) {

if ((nread = rio\_read(rp, bufp, nleft)) < 0) {

return -1; /\* errno встановлюється read() \*/

} else if (nread == 0) {

break; /\* EOF \*/

}

nleft -= (size\_t)nread;

bufp += nread;

}

return (ssize\_t)(n - nleft); /\* return >= 0 \*/

}

/\*

\* rio\_readlineb - Надійне зчитування текстової строки

\*/

ssize\_t rio\_readlineb(rio\_t \*rp, void \*usrbuf, size\_t maxlen) {

size\_t n;

ssize\_t rc;

char c, \*bufp = usrbuf;

for (n = 1; n < maxlen; n++) {

if ((rc = rio\_read(rp, &c, 1)) == 1) {

\*bufp++ = c;

if (c == '\n') {

n++;

break;

}

} else if (rc == 0) {

if (n == 1) {

return 0; /\* EOF, ніякої інформації не було прочитано \*/

} else {

break; /\* EOF, частина інформації була прочитана \*/

}

} else {

return -1; /\* Помилка \*/

}

}

\*bufp = 0;

return (ssize\_t)(n - 1);

}

**helper.h**

/\*

\* - Пакет RIO (надійний ввод-вивод), який дозволяє виконувати зчитування

\* та запис надійно, обробляючи короткі зчитування та записи.

\* Він також надає rio\_t, який дозволяє виконувати буферизоване зчитування.

\*/

#ifndef HELPER\_H

#define HELPER\_H

#include<stdlib.h>

#include<stdio.h>

#define RIO\_BUFSIZE 8192

typedef struct {

int rio\_fd; /\* Дескриптор для цього внутрішнього buf \*/

ssize\_t rio\_cnt; /\* Непрочитані байти у внутрішньому buf \*/

char \*rio\_bufptr; /\* Наступний непрочитаний байт у внутрішньому buf \*/

char rio\_buf[RIO\_BUFSIZE]; /\* Внутрішній буфер \*/

} rio\_t;

/\* Rio (Robust I/O) пакет \*/

ssize\_t rio\_readn(int fd, void \*usrbuf, size\_t n);

ssize\_t rio\_writen(int fd, const void \*usrbuf, size\_t n);

void rio\_readinitb(rio\_t \*rp, int fd);

ssize\_t rio\_readnb(rio\_t \*rp, void \*usrbuf, size\_t n);

ssize\_t rio\_readlineb(rio\_t \*rp, void \*usrbuf, size\_t maxlen);

#endif