服务端server.c：

#include <ctype.h>

#include <errno.h>

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define BACKLOG 25

#define COMMAND\_LENGTH 24

#define MAX\_CONNECTIONS 20

#define MESSAGE\_LENGTH\_LIMIT 255

#define PORT "1337"

#define CMD\_NONE 0

#define CMD\_QUIT 1

void caesar(char \*str)

{

int ndx;

char chr, offset;

for (ndx = 0; ndx < strlen(str); ndx++) {

chr = str[ndx];

if (isalpha(chr)) {

offset = isupper(chr) ? 'A': 'a';

chr = chr - offset + 3;

chr %= 26;

chr = chr + offset;

str[ndx] = chr;

}

}

}

int handle\_message(int connfd)

{

char buffer[MESSAGE\_LENGTH\_LIMIT];

int rv;

memset(buffer, 0, MESSAGE\_LENGTH\_LIMIT);

rv = recv(connfd, buffer, MESSAGE\_LENGTH\_LIMIT - 1, 0);

if (rv == 0) {

fprintf(stderr, "Remote host hung up\n");

return -1;

} else if (rv == -1) {

perror("server: recv");

return -1;

} else {

printf("server: recv %3d: %s", rv, buffer);

caesar(buffer);

if ((rv = send(connfd, buffer, strlen(buffer), 0)) == -1) {

perror("server: send");

return -1;

} else {

printf("server: send %3d: %s", rv, buffer);

}

}

return 0;

}

int handle\_connection(sockfd)

{

int connfd;

struct sockaddr\_storage remote\_addr;

unsigned int addr\_size = sizeof(remote\_addr);

if ((connfd = accept(sockfd, (struct sockaddr \*) &remote\_addr, &addr\_size))

== -1) {

fprintf(stderr, "Could not accpet: %s\n", strerror(errno));

return -1;

}

return connfd;

}

int handle\_command(int fd)

{

char buffer[COMMAND\_LENGTH + 1];

memset(buffer, 0, COMMAND\_LENGTH + 1);

fgets(buffer, COMMAND\_LENGTH, stdin);

if (strcmp(buffer, "quit\n") == 0)

return CMD\_QUIT;

return CMD\_NONE;

}

void run\_server(int sockfd)

{

fd\_set readfds;

int conn\_fds[MAX\_CONNECTIONS], readyfds, ndx, done = 0, newfd;

int max\_fd = sockfd;

// Set every fd to -1.

for (ndx = 0; ndx < MAX\_CONNECTIONS; ndx++)

conn\_fds[ndx] = -1;

// Add stdin and our listening socket to file descriptor list.

conn\_fds[0] = STDIN\_FILENO;

conn\_fds[1] = sockfd;

while (!done) {

FD\_ZERO(&readfds);

for (ndx = 0; ndx < MAX\_CONNECTIONS; ndx++) {

if (conn\_fds[ndx] != -1)

FD\_SET(conn\_fds[ndx], &readfds);

}

readyfds = select(max\_fd + 1, &readfds, NULL, NULL, NULL);

if (readyfds == -1)

perror("server: select");

for (ndx = 0; ndx < MAX\_CONNECTIONS; ndx++) {

if (conn\_fds[ndx] == -1)

continue;

if (FD\_ISSET(conn\_fds[ndx], &readfds)) {

// We recieved a command message on stdin.

if (conn\_fds[ndx] == STDIN\_FILENO) {

switch (handle\_command(STDIN\_FILENO)) {

case CMD\_QUIT:

printf("quitting...\n");

for (ndx = 2; ndx < MAX\_CONNECTIONS; ndx++) {

if (conn\_fds[ndx] != -1)

close(conn\_fds[ndx]);

}

done = 1;

break;

}

break;

// A new connection is ready on the listening socket.

} else if (conn\_fds[ndx] == sockfd) {

if ((newfd = handle\_connection(sockfd)) != -1) {

for (ndx = 0; ndx < MAX\_CONNECTIONS; ndx++) {

if (conn\_fds[ndx] == -1) {

conn\_fds[ndx] = newfd;

break;

}

}

if (newfd > max\_fd)

max\_fd = newfd;

fprintf(stderr, "fromed new connection\n");

}

break;

// A connection is ready with data.

} else {

if (handle\_message(conn\_fds[ndx]) == -1) {

close(conn\_fds[ndx]);

conn\_fds[ndx] = -1;

}

break;

}

}

}

}

}

int main(void)

{

int sockfd, rv;

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

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_flags = AI\_PASSIVE;

if ((rv = getaddrinfo(NULL, PORT, &hints, &servinfo)) != 0) {

fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(rv));

return 1;

}

for (p = servinfo; p != NULL; p = p->ai\_next) {

if ((sockfd = socket(p->ai\_family, p->ai\_socktype,

p->ai\_protocol)) == -1) {

perror("server: socket");

continue;

}

if (bind(sockfd, p->ai\_addr, p->ai\_addrlen) == -1) {

close(sockfd);

perror("server: bind");

continue;

}

break;

}

if (p == NULL) {

fprintf(stderr, "server: failed to bind\n");

return 2;

}

freeaddrinfo(servinfo);

if (listen(sockfd, BACKLOG) == -1) {

perror("listen");

exit(1);

}

run\_server(sockfd);

close(sockfd);

return 0;

}

客户端client.c：

#include <errno.h>

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define MESSAGE\_LENGTH\_LIMIT 255

#define PORT "1337"

int send\_message(int sockfd)

{

char buffer[MESSAGE\_LENGTH\_LIMIT];

int rv;

memset(buffer, 0, MESSAGE\_LENGTH\_LIMIT);

fgets(buffer, MESSAGE\_LENGTH\_LIMIT, stdin);

if ((rv = send(sockfd, buffer, strlen(buffer), 0)) == -1) {

perror("client: send");

return -1;

} else {

printf("client: send %3d: %s", rv, buffer);

if ((rv = recv(sockfd, buffer, MESSAGE\_LENGTH\_LIMIT - 1, 0)) <= 0) {

if (rv == 0)

fprintf(stderr, "connection closed by peer\n");

else

perror("client: recv");

return -1;

} else {

printf("client: recv %3d: %s", rv, buffer);

}

}

return 0;

}

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

{

int sockfd, rv;

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

if (argc != 2) {

fprintf(stderr, "usage: client hostname\n");

exit(1);

}

memset(&hints, 0, sizeof hints);

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

if ((rv = getaddrinfo(argv[1], PORT, &hints, &servinfo)) != 0) {

fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(rv));

return 1;

}

for (p = servinfo; p != NULL; p = p->ai\_next) {

if ((sockfd = socket(p->ai\_family, p->ai\_socktype,

p->ai\_protocol)) == 1) {

perror("client: socket");

continue;

}

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

close(sockfd);

perror("client: connect");

continue;

}

break;

}

if (p == NULL) {

fprintf(stderr, "client: failed to connect\n");

return 2;

}

while (send\_message(sockfd) != -1)

;

close(sockfd);

return 0;

}