EE 450

Lab 2

Report

Name: Yirou Qian

Student ID: 4354716429

Part 1. The Whole Code

1. Server.cpp

#include "Server.hpp"

#include <cstdio>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <sys/wait.h>

#include <signal.h>

#include <string>

#include <iostream>

#include <sstream>

#define PORT "3729" *// 3300 + 429 (USC ID: 4354 7164 29)*

#define BACKLOG 10 *// how many pending connections queue will hold*

#define MAXDATASIZE 1024 *// max number of bytes*

**using** **namespace** std;

**void** sigchld\_handler(**int** s)

{

**while**(waitpid(-1, **NULL**, WNOHANG) > 0);

}

*// get sockaddr, IPv4 or IPv6:*

**void** \*get\_in\_addr(**struct** sockaddr \*sa)

{

**if** (sa->sa\_family == AF\_INET) {

**return** &(((**struct** sockaddr\_in\*)sa)->sin\_addr);

}

**return** &(((**struct** sockaddr\_in6\*)sa)->sin6\_addr);

}

string int\_to\_string (**int** num) {

**char** tmp[15] = {0};

snprintf(tmp, 15, "%d", num);

**return** tmp;

}

string random\_transaction\_ID () {

**int** id = rand() % 255;

**return** int\_to\_string (id);

}

string random\_IP\_address (string randID) {

string address = "";

**int** tmp = 0;

**for** (**int** i = 0; i < 3; i++) {

tmp = rand() % 255;

address = int\_to\_string (tmp) + "." + address;

}

**return** address + randID;

}

**int** main(**void**) {

**int** sockfd, new\_fd; *// listen on sock\_fd, new connection on new\_fd*

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

**struct** sockaddr\_storage their\_addr; *// connector's address information*

socklen\_t sin\_size;

**struct** sigaction sa;

**int** yes=1;

**char** s[INET6\_ADDRSTRLEN];

**char** buffer[MAXDATASIZE];

**int** rv;

**int** numbytes;

memset(&hints, 0, **sizeof** hints);

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_flags = AI\_PASSIVE; *// use my IP*

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

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

**return** 1;

}

*// loop through all the results and bind to the first we can*

**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** (setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &yes,

**sizeof**(**int**)) == -1) {

perror("setsockopt");

exit(1);

}

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

close(sockfd);

perror("server: bind");

**continue**;

}

**break**;

}

freeaddrinfo(servinfo); *// all done with this structure*

**if** (p == **NULL**) {

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

exit(1);

}

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

perror("listen");

exit(1);

}

sa.sa\_handler = sigchld\_handler; *// reap all dead processes*

sigemptyset(&sa.sa\_mask);

sa.sa\_flags = SA\_RESTART;

**if** (sigaction(SIGCHLD, &sa, **NULL**) == -1) {

perror("sigaction");

exit(1);

}

printf("server: waiting for connections...\n");

**while**(1) { *// main accept() loop*

sin\_size = **sizeof** their\_addr;

new\_fd = accept(sockfd, (**struct** sockaddr \*)&their\_addr, &sin\_size);

**if** (new\_fd == -1) {

perror("accept");

**continue**;

}

**struct** sockaddr \*sa = (**struct** sockaddr \*)&their\_addr;

**if** (sa->sa\_family == AF\_INET)

{

inet\_ntop(their\_addr.ss\_family, &(((**struct** sockaddr\_in\*)sa)->sin\_addr), s, **sizeof** s);

}

**else**

{

inet\_ntop(their\_addr.ss\_family, &(((**struct** sockaddr\_in6\*)sa)->sin6\_addr), s, **sizeof** s);

}

*//inet\_ntop(their\_addr.ss\_family, get\_in\_addr((struct sockaddr \*)&their\_addr), s, sizeof s);*

printf("server: got connection from %s\n", s);

**if** (!fork()) { *// this is the child process*

close(sockfd);*// child doesn't need the listener*

*// receive requests from the client*

**if** ((numbytes = recv(new\_fd, buffer, MAXDATASIZE-1, 0)) > 0) {

printf("server: receive transaction ID from client: %s\n", buffer);

}

*// offer phase*

string strAddr = random\_IP\_address(buffer);

string strID = random\_transaction\_ID();

string strMsg = strAddr + "#" + strID;

strcpy(buffer, strMsg.c\_str());

**if** ((numbytes = send(new\_fd, buffer , MAXDATASIZE-1, 0)) > 0) {

printf("server: offer IP address to client: %s\n", strAddr.c\_str());

printf("server: generate new transaction ID: %s\n", strID.c\_str());

}

*// acknlowledge phase*

**if** ((numbytes = recv(new\_fd, buffer, MAXDATASIZE-1, 0)) > 0) {

printf("server: acknowledge with new transaction ID: %s\n", buffer);

}

strID = random\_transaction\_ID();

strcpy(buffer, strID.c\_str());

**if** ((numbytes = send(new\_fd, buffer, MAXDATASIZE-1, 0)) == -1) {

perror("send");

}

printf("server: send ACK and IP address to client: %s\n", strAddr.c\_str());

printf("server: generate new transaction ID: %s\n", buffer);

close(new\_fd);

exit(0);

}

close(new\_fd); *// parent doesn't need this*

}

**return** 0;

}

1. Client.cpp

#include "Client.hpp"

#include <cstdio>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <cstring>

#include <netdb.h>

#include <sys/types.h>

#include <netinet/in.h>

#include <sys/socket.h>

#include <string>

#include <iostream>

#include <sstream>

#include <arpa/inet.h>

#define PORT "3729" *// 3300 + 429 (USC ID: 4354 7164 29)*

#define MAXDATASIZE 1024 *// max number of bytes*

**using** **namespace** std;

*// get sockaddr, IPv4 or IPv6:*

**void** \*get\_in\_addr(**struct** sockaddr \*sa)

{

**if** (sa->sa\_family == AF\_INET) {

**return** &(((**struct** sockaddr\_in\*)sa)->sin\_addr);

}

**return** &(((**struct** sockaddr\_in6\*)sa)->sin6\_addr);

}

string int\_to\_string (**int** num) {

**char** tmp[15] = {0};

snprintf(tmp, 15, "%d", num);

**return** tmp;

}

string random\_transaction\_ID () {

**int** id = rand() % 255;

**return** int\_to\_string (id);

}

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

srand(time(**NULL**));

**int** sockfd, numbytes;

**char** buffer[MAXDATASIZE];

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

**int** rv;

**char** s[INET6\_ADDRSTRLEN];

**if** (argc != 2) {

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

exit(1);

}

memset(&hints, 0, **sizeof** hints);

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

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

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

**return** 1;

}

*// loop through all the results and connect to the first we can*

**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;

}

**struct** sockaddr \*sa = (**struct** sockaddr \*)p->ai\_addr;

**if** (sa->sa\_family == AF\_INET)

{

inet\_ntop(p->ai\_family, &(((**struct** sockaddr\_in\*)sa)->sin\_addr), s, **sizeof** s);

}

**else**

{

inet\_ntop(p->ai\_family, &(((**struct** sockaddr\_in6\*)sa)->sin6\_addr), s, **sizeof** s);

}

*//inet\_ntop(p->ai\_family, get\_in\_addr((struct sockaddr \*)p->ai\_addr), s, sizeof s);*

printf("client: connecting to %s\n", s);

freeaddrinfo(servinfo); *// all done with this structure*

*// discover phase*

string strID = random\_transaction\_ID();

strcpy(buffer, strID.c\_str());

**if** ((numbytes = send(sockfd, buffer, MAXDATASIZE-1, 0)) > 0) {

printf("client: generate transaction ID: %s\n", buffer);

}

*//receive IP address from server*

string strRecvAddr;

string strRecvID;

**if** ((numbytes = recv(sockfd, buffer, MAXDATASIZE-1, 0)) > 0) {

string msg = string(buffer);

**int** divideAddr = msg.find('#');

strRecvAddr = msg.substr(0, divideAddr);

strRecvID = msg.substr(divideAddr+1);

printf("client: get IP address from server: %s\n", strRecvAddr.c\_str());

printf("client: generate new transaction ID: %s\n", strRecvID.c\_str());

}

*// request phase*

strID = random\_transaction\_ID();

strcpy(buffer, strID.c\_str());

**if** ((numbytes = send(sockfd, buffer, MAXDATASIZE-1, 0)) > 0) {

printf("client: want to take this IP address from server: %s\n", strRecvAddr.c\_str());

printf("client: generate new transaction ID: %s\n", buffer);

}

**if** ((numbytes = recv(sockfd, buffer, MAXDATASIZE-1, 0)) > 0) {

printf("client: successfully get IP address: %s\n", strRecvAddr.c\_str());

}

**if** ((numbytes = recv(sockfd, buffer, MAXDATASIZE-1, 0)) == -1) {

perror("recv");

exit(1);

}

buffer[numbytes] = '\0';

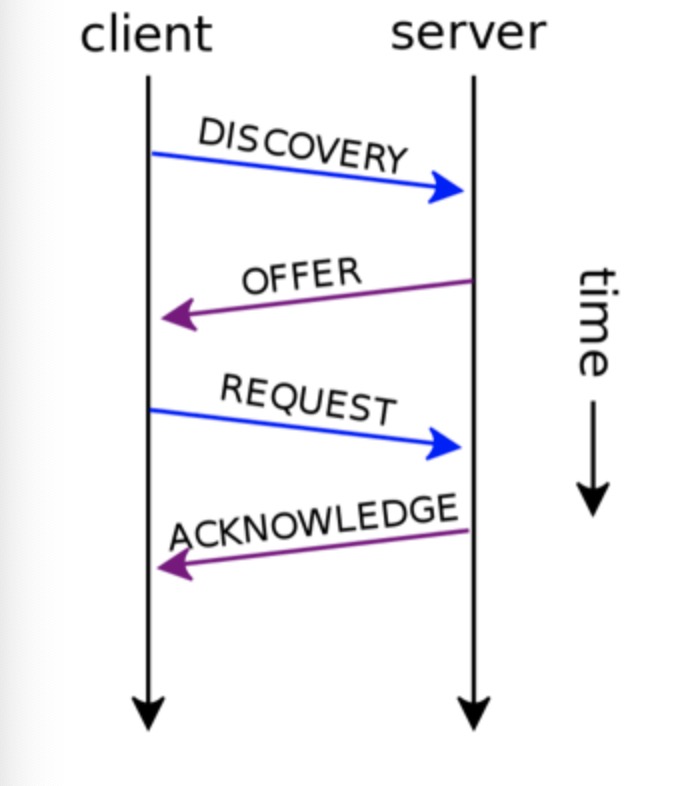
close(sockfd);

**return** 0;

}

Part 2. Key Aspects of Implementation

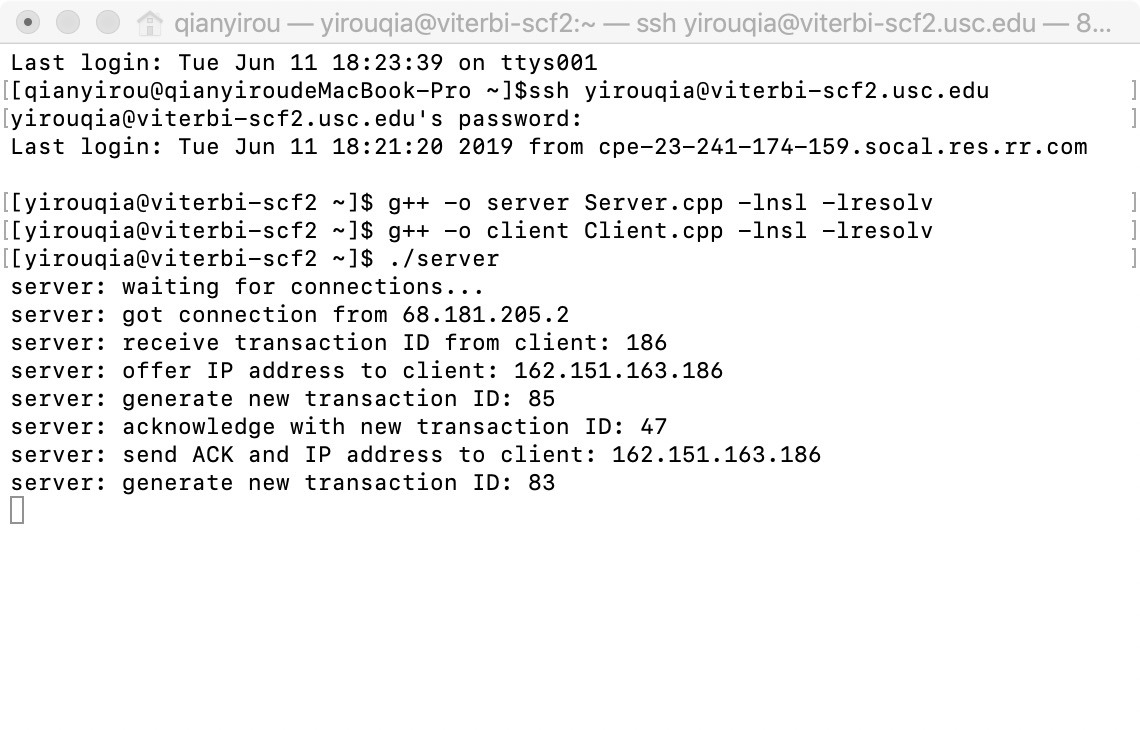
There are four phases of the whole DHCP operations: Discovery, Offer, Request, and ACK.



1. Discovery: First, the client should randomly generate a transaction ID, which is 8 bits and its range should be in 0~255. Then send it to server, and at that moment server should have prepared for such connection.
2. Offer: After server receives the transaction ID from the client, server will generate a random IPv4 address with 32 bits. In the 32 bits, the front 24 bits are randomly; however, the final 8 bits are the transaction ID received from client before. Also, server will generate another new transaction ID. Then server will send IP address and transaction ID to client together.
3. Request: Client will receive the IP address and transaction ID from server. Then client thinks this IP address is good and it want to take this IP address as its own address. As a result, client will echo this IP address which means that the client wants to take it. When client sends this request, it will also generate another new transaction ID.
4. Acknowledge: After receiving the request from client, server will confirm this request and send acknowledge information, like “ACK”, together with another new transaction ID to client. It means that now client can use this IP address.

Part 3. Experimental Results and The Output File Format

1. Server.cpp



1. Client.cpp

