Lab 6: Network Chat
CMSC 242
Due: April 22, 2016 by 11:59:59pm

Network chat is a typical example of the use of sockets to implement a client/server application. The client connects to the server and relays everything the user types over the remote connection. It also listens for incoming network traffic and displays it on the screen. The server listens for incoming connections. It broadcasts any incoming traffic from a client to all the other clients.

Step 0. Setting Up

Create a folder named Lab6. In that folder, create the following Makefile:

```
all: server client

server: server.c
	gcc -g server.c -o Server

client: client.c
	gcc -g client.c -o Client

.phony: clean

clean:
	rm -f Server Client *.o
```

Step 1. Server

Create a file named “server.c”. In that file, type:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/types.h>

struct server_state {
	int listen_socket;
	int client_fds[40];
	int num_clients;
	int maxfd;
	fd_set monitor_fds;
};

int setup_listen_socket() {
	int sockfd;
	struct sockaddr_in list_addr;
	int result;

	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd < 0) {
		perror("socket");
		exit(-1);
	}
	printf("Binding socket.\n");
	list_addr.sin_family = AF_INET;
	inet_pton(AF_INET, "0.0.0.0", &list_addr.sin_addr.s_addr);
	list_addr.sin_port = htons(4000);
	result = bind(sockfd, (struct sockaddr *)&list_addr, sizeof(struct sockaddr_in));
	if (result < 0) {
		perror("bind");
		exit(-1);
	}

	printf("Listening.\n");
```

```
        result = listen(sockfd,1);

        if (result < 0) {
                perror("listen");
                exit(-1);
        }
        return sockfd;
}

int connect_client(struct server_state* state) {
        int client_sock;
        struct sockaddr_in client;
        int result;
        socklen_t addrlen;

        //Too many clients
        if (state->num_clients >=40)
                return -1;

        client_sock = accept(state->listen_socket, (struct sockaddr *)&client, &addrlen);

        //Accept error
        if (client_sock < 0) {
                perror("accept");
                return -1;
        }

        if (client_sock > state->maxfd)
                state->maxfd=client_sock;

        FD_SET(client_sock, &state->monitor_fds);

        char buf[1024];
        strncpy(buf, "Welcome to network chat.\n", 25);
        buf[26]='\0';
        result = write(client_sock, buf, 1024);
        if (result < 0) {
                perror("write");
                exit(-1);
        }

        state->client_fds[state->num_clients] = client_sock;
        state->num_clients++;
        return 1;
}

int main(int argc, char* argv[])
{
        int result;
        char cmd;

        int i, j;

        struct server_state state;

        fd_set read_fds;
        fd_set except_fds;

        state.num_clients = 0;
        state.listen_socket = setup_listen_socket();

        FD_ZERO(&state.monitor_fds);
        FD_SET(0, &state.monitor_fds);
        FD_SET(state.listen_socket, &state.monitor_fds);
        state.maxfd = state.listen_socket;
        cmd = ' ';

        printf("Server ready.  Type 'q' to quit.\n");
        while (cmd != 'q') {
                read_fds = state.monitor_fds;
                except_fds = state.monitor_fds;
                result = select(state.maxfd + 1, &read_fds, NULL, &except_fds, NULL);
                if (FD_ISSET(state.listen_socket, &read_fds)) {
                        result = connect_client(&state);
                }
                if (FD_ISSET(0, &read_fds)) {
                        cmd = getc(stdin);
```

```
                }
                for (i=0;i<state.num_clients; ++i) {
                        if (FD_ISSET(state.client_fds[i], &read_fds)) {
                                char buf[1024];
                                result = read(state.client_fds[i], buf, 1024);
                                if (result <= 0) {
                                        printf("Client %d disconnected.\n", i);
                                        FD_CLR(state.client_fds[i], &state.monitor_fds);
                                        close(state.client_fds[i]);
                                        state.num_clients--;
                                        state.client_fds[i] = state.client_fds[state.num_clients];
                                }
                                else{
                                        printf("Client %d: %s\n", i, buf);
                                        for (j=0; j < state.num_clients; ++j) {
                                                if (i == j)
                                                        continue;
                                                result = write(state.client_fds[j], buf, 1024);
                                                if (result <= 0) {
                                                        perror("write");
                                                        exit(-1);
                                                }
                                        }
                                }
                        }
                        if (FD_ISSET(state.client_fds[i], &except_fds)) {
                                        printf("Client %d disconnected.\n", i);
                                        FD_CLR(state.client_fds[i], &state.monitor_fds);
                        }
                }
        }

        for (i=0; i < state.num_clients; ++i)
                close(state.client_fds[i]);
        close(state.listen_socket);
}
```

Step 2. Client

Now make a file named “client.c”. In that file, type:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc, char* argv[])
{
        int sockfd;
        int result;
        fd_set listen_fds;
        fd_set read_fds;
        fd_set except_fds;
        struct sockaddr_in addr;
        char buf[1024];

        if (argc < 2) {
                printf("Syntax: Client <ip address>\n");
                exit(-1);
        }

        //Create a network socket
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd < 0) {
                perror("socket");
                exit(-1);
        }

        addr.sin_family = AF_INET;
        addr.sin_port = htons(4000);
```

```
        inet_pton(AF_INET, argv[1], &addr.sin_addr.s_addr);

        printf("Connecting...\n");
        result = connect(sockfd, (struct sockaddr *)&addr, sizeof(struct sockaddr_in));

        if (result < 0 ) {
                perror("connect");
                exit(-1);
        }

        printf("Connected.\n");

        FD_ZERO(&listen_fds);
        FD_SET(0, &listen_fds);
        FD_SET(sockfd, &listen_fds);

        buf[0] = '\0';
        while (strncmp(buf, "quit", 4) != 0) {
                read_fds = listen_fds;
                result = select(sockfd+1, &read_fds, NULL, &except_fds, NULL);
                if (FD_ISSET(0, &read_fds)){
                        fgets(buf, 1024, stdin);
                        write(sockfd, buf, 1024);
                }
                else {
                        result = read(sockfd, buf, 1024);
                        if (result <= 0) {
                                printf("Connection terminated.\n");
                                printf("%s", buf);
                                strncpy(buf, "quit", 4);
                        }
                        else {
                                printf("%s", buf);
                                buf[0] = '\0';
                        }
                }
        }
        close(sockfd);
}
```

Test your code by typing “./Server” in one terminal and then typing “./Client 127.0.0.1” in another terminal. Open up additional clients in other terminals.

Step 3. Questions

In a file named “Questions.txt”, answer the following questions:

1. What port does the network server run on?

2. Does the chat program use TCP or UDP? How do you know?

3. How many simultaneous clients are allowed connect to the server?

4. What does the “connect” function call in the client do?

5. The address passed to inet_pton in the server is “0.0.0.0”. Why can we connect to “127.0.0.1”?

6. The client takes an IP address. What system call would we use to allow it to take a hostname?

7. This server runs “single-threaded” – there is no forking or spawning of new threads. Why is that probably not a good way to design the program?

Submitting
As usual, create a tarball of your entire Lab6 folder and submit through the course web site.