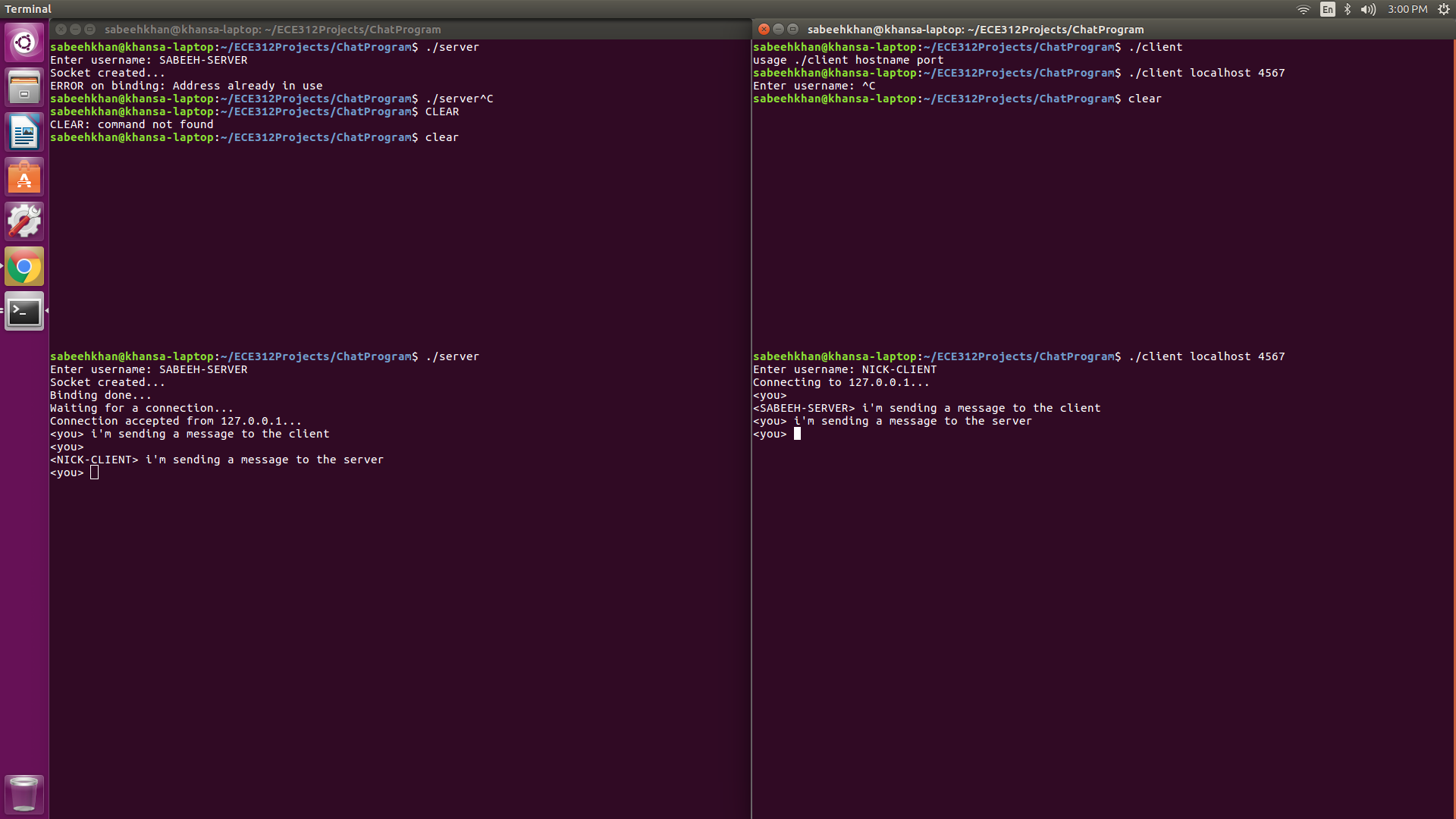
**Sabeeh Khan and Nick Kiesel**

**ECE 312 – 01**

**Project 1: Socket Chat Program**

**Figure 1: Screenshot of Server/Client Chat Operation**

****

*Left = server, right = client*

**Server.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <pthread.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 4567

#define BUF\_SIZE 256

#define CLADDR\_LEN 100

int nClose;

void error(char \*msg)

{

perror(msg);

exit(1);

}

void \* receiveMessage(void \* socket) {

int sockfd, ret;

char buffer[BUF\_SIZE], send\_user[BUF\_SIZE];

sockfd = (int) socket;

int wasMe = 0;

memset(send\_user, 0, BUF\_SIZE);

ret = read(sockfd, send\_user, BUF\_SIZE);

if (ret < 0)

printf("Error receiving data!\n");

memset(buffer, 0, BUF\_SIZE);

while (nClose) {

ret = read(sockfd, buffer, BUF\_SIZE);

if (strcmp(buffer,"exit\n") == 0) {

nClose = 0;

wasMe = 1;

}

if (ret > 0) {

printf("\n<%s> %s<you> ", send\_user, buffer);

fflush(stdout);

}

memset(buffer, 0, BUF\_SIZE);

}

if (ret < 0)

printf("Error receiving data!\n");

if (wasMe)

printf("Chat ended. Press enter to exit\n");

close(sockfd);

}

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

{

int sockfd, newsockfd, portno, clilen;

char buffer[BUF\_SIZE], user[BUF\_SIZE];

struct sockaddr\_in serv\_addr, cli\_addr;

pid\_t childpid;

char clientAddr[CLADDR\_LEN];

pthread\_t rThread;

int ret, n;

nClose = 1;

printf("Enter username: ");

fgets(user,255,stdin);

strtok(user,"\n");

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0)

error("ERROR opening socket");

printf("Socket created...\n");

memset(&serv\_addr, 0, sizeof(serv\_addr));

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

serv\_addr.sin\_port = htons(PORT);

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(sockfd, (struct sockaddr \*) &serv\_addr, sizeof(serv\_addr)) < 0)

error("ERROR on binding");

printf("Binding done...\n");

printf("Waiting for a connection...\n");

listen(sockfd,5);

clilen = sizeof(cli\_addr);

newsockfd = accept(sockfd, (struct sockaddr \*) &cli\_addr, &clilen);

inet\_ntop(AF\_INET, &(cli\_addr.sin\_addr), clientAddr, CLADDR\_LEN);

printf("Connection accepted from %s...\n", clientAddr);

//creating a new thread for receiving messages from the client

if (ret = pthread\_create(&rThread, NULL, receiveMessage, (void \*) newsockfd)) {

printf("ERROR: Return Code from pthread\_create() is %d\n", ret);

error("ERROR creating thread");

}

n = write(newsockfd,user,strlen(user));

if (n < 0) {

error("ERROR writing to socket");

}

while(nClose){

bzero(buffer,256);

printf("<you> ");

fgets(buffer,255,stdin);

if (nClose) {

n = write(newsockfd,buffer,strlen(buffer));

if (n < 0) {

error("ERROR writing to socket");

break;

}

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

nClose = 0;

}

}

if (newsockfd < 0)

error("ERROR on accept");

// close(newsockfd);

close(sockfd);

pthread\_exit(NULL);

return 0;

}

**Client.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <pthread.h>

#define BUF\_SIZE 256

#define SERVADDR\_LEN 100

int nClose;

char username[256] = { 0 };

void error(const char \*msg)

{

perror(msg);

exit(0);

}

void \* receiveMessage(void \* socket) {

int sockfd, ret;

char buffer[BUF\_SIZE],send\_user[BUF\_SIZE];

sockfd = (int) socket;

int wasMe = 0;

memset(send\_user, 0, BUF\_SIZE);

ret = read(sockfd, send\_user, BUF\_SIZE);

if (ret < 0)

printf("Error receiving data!\n");

memset(buffer, 0, BUF\_SIZE);

while (nClose) {

ret = read(sockfd, buffer, BUF\_SIZE);

if (strcmp(buffer,"exit\n") == 0) {

nClose = 0;

wasMe = 1;

}

if (ret > 0) {

printf("\n<%s> %s<you> ", send\_user, buffer);

fflush(stdout);

}

memset(buffer, 0, BUF\_SIZE);

}

if (ret < 0)

printf("Error receiving data!\n");

if (wasMe)

printf("Chat ended. Press enter to exit\n");

close(sockfd);

}

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

{

int sockfd, portno, n, ret;

pthread\_t rThread;

struct sockaddr\_in serv\_addr;

struct hostent \*server;

char user[BUF\_SIZE], servAddr[SERVADDR\_LEN];

nClose = 1;

char buffer[256];

if (argc < 3) {

fprintf(stderr,"usage %s hostname port\n", argv[0]);

exit(0);

}

printf("Enter username: ");

fgets(user,255,stdin);

strtok(user,"\n");

portno = atoi(argv[2]);

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0)

error("ERROR opening socket");

server = gethostbyname(argv[1]);

if (server == NULL) {

fprintf(stderr,"ERROR, no such host\n");

exit(0);

}

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

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

bcopy((char \*)server->h\_addr,

(char \*)&serv\_addr.sin\_addr.s\_addr,

server->h\_length);

serv\_addr.sin\_port = htons(portno);

if (connect(sockfd,(struct sockaddr \*) &serv\_addr,sizeof(serv\_addr)) < 0)

error("ERROR connecting");

inet\_ntop(AF\_INET, &(serv\_addr.sin\_addr), servAddr, SERVADDR\_LEN);

printf("Connecting to %s...\n", servAddr);

//creating a new thread for receiving messages from the client

if (ret = pthread\_create(&rThread, NULL, receiveMessage, (void \*) sockfd)) {

printf("ERROR: Return Code from pthread\_create() is %d\n", ret);

error("ERROR creating thread");

}

n = write(sockfd,user,strlen(user));

if (n < 0) {

error("ERROR writing to socket");

}

while(nClose){

bzero(buffer,256);

printf("<you> ");

fgets(buffer,255,stdin);

if (nClose) {

n = write(sockfd,buffer,strlen(buffer));

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

nClose = 0;

if (n < 0) {

error("ERROR writing to socket");

break;

}

}

}

close(sockfd);

return 0;

}

**Response**

We would implement an actual group chat by first having the server acquire a list of all connected clients. One of the clients would send their username and message to the server. The server would then identify which client it received the message from and send that username and message to all other clients. Each client will have two threads for receiving and transmitting messages. The server will have three threads: two for receiving and transmitting messages, the third for establishing new connections.