CS232 Operating Systems
Assignment 04: Introduction to Socket Programming

Mudasir Hanif Shaikh (ms03831), Kainat Abbasi (ka04051)

Fall 2019

1 Client

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <netdb.h>

#define BUF_SIZE 4096

void * threadedInput(void * socket)
{
int sock = *((int *) socket);
char *inputBuffer;
int inputBufferSize = 120;

inputBuffer = (char *)malloc(inputBufferSize * sizeof(char));

while(1){
if (getline(&inputBuffer, (size_t*) &inputBufferSize, stdin) != 0)
    {
//printf("%s\n", inputBuffer);
if(write(sock, inputBuffer, strlen(inputBuffer)) < 0) perror("send"
    );
}
}
free(inputBuffer);
return NULL;
}

int main(int argc, char* argv[]){
if (argc != 4){
printf("%s\n", "please pass appropriate arguments... exiting");
exit(1);
}

char * hostname = argv[1];     //the hostname we are looking up
short port = atoi(argv[2]);                    //the port we are connecting on

struct addrinfo *result;       //to store results
struct addrinfo hints;         //to indicate information we want

struct sockaddr_in *saddr_in;  //socket interent address

int s, n;                       //for error checking

int client_socket;                          //socket file descriptor

pthread_t client_thread;

char* client_name = argv[3];

char response[BUF_SIZE];           //read in 4096 byte chunks

//setup our hints
memset(&hints, 0, sizeof(struct addrinfo));  //zero out hints
hints.ai_family = AF_INET; //we only want IPv4 addresses

//Convert the hostname to an address
if( (s = getaddrinfo(hostname, NULL, &hints, &result)) != 0){
fprintf(stderr, "getaddrinfo: %s\n",gai_strerror(s));
exit(1);
}

//convert generic socket address to inet socket address
saddr_in = (struct sockaddr_in *) result->ai_addr;

//set the port in network byte order
saddr_in->sin_port = htons(port);

//open a socket
if( (client_socket = socket(AF_INET, SOCK_STREAM, 0))  < 0){
perror("socket");
exit(1);
}

//connect to the server
if( connect(client_socket, (struct sockaddr *) saddr_in, sizeof(*saddr_in)) < 0 ) {
perror("connect");
exit(1);
}

//send the client_name
if( write(client_socket, client_name, strlen(client_name)) < 0 ){
perror("send");
}

//read the response until EOF
pthread_create(&client_thread, NULL, threadedInput, &client_socket)
```

```
    ;

while (1)
{
memset(response, 0, BUF_SIZE);
n = read(client_socket, response, BUF_SIZE-1);
if(n <= 0){ //close the socket
close(client_socket);
printf("Socket Closed %d\n:", client_socket);
exit(1);
return 0;
}
else {
printf("%s", response);
}
}
return 0; //success
}
```

2 Server

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <netdb.h>
#include <semaphore.h>


#define BUF_SIZE 4096

char* listCommand = "/list\n";

char* msgCommand = "/msg";

char* quitCommand = "/quit";



pthread_t clientThread;

typedef struct __node {
char* clientName;
int clientSock; //file descriptor
struct __node* next;
} myNode;

myNode* linkedList = NULL; //head

typedef struct _rwlock_t {
sem_t lock; // binary semaphore (basic lock)
sem_t writelock; // allow ONE writer/MANY readers
int readers; // #readers in critical section
```

```
} rwlock_t;

rwlock_t lock;

void rwlock_init(rwlock_t *rw) {
rw->readers = 0;
sem_init(&rw->lock, 0, 1);
sem_init(&rw->writelock, 0, 1);
}

void rwlock_acquire_readlock(rwlock_t *rw) {
sem_wait(&rw->lock);
rw->readers++;
if (rw->readers == 1) // first reader gets writelock
sem_wait(&rw->writelock);
sem_post(&rw->lock);
}

void rwlock_release_readlock(rwlock_t *rw) {
sem_wait(&rw->lock);
rw->readers--;
if (rw->readers == 0) // last reader lets it go
sem_post(&rw->writelock);
sem_post(&rw->lock);
}

void rwlock_acquire_writelock(rwlock_t *rw) {
sem_wait(&rw->writelock);
}

void rwlock_release_writelock(rwlock_t *rw) {
sem_post(&rw->writelock);
}


void enque(myNode* new){
rwlock_acquire_writelock(&lock);
myNode* temp = linkedList;
if (linkedList == NULL){
linkedList = new;
linkedList->next = NULL;
rwlock_release_writelock(&lock);
return;
}
else {
new->next = linkedList;
new->next->next = temp->next;
}
linkedList = new;
rwlock_release_writelock(&lock);
}

int deque(int client){ //remove client having file descriptor
    client
rwlock_acquire_writelock(&lock);
myNode* current = linkedList;
myNode* previous = linkedList;
```

```

//int flag = 0;

if (linkedList == NULL) { //if linked list is empty, release lock
    and return
rwlock_release_writelock(&lock);
return 0;
}

if (linkedList->clientSock == client){
linkedList = linkedList->next;
printf("Client %i Removed.....\n", client);
free(current->clientName);
free(current);
rwlock_release_writelock(&lock);
return client; //just to check if the client that we removed is
    infact the client that we wanted to remove
}

while (current->next != NULL && current->clientSock != client){
previous = current;
current = current->next;
}
if (current != NULL){
if (current->clientSock == client){
printf("Client %i Removed.....\n", client);
previous->next = current->next;
free(current->clientName);
free(current);
rwlock_release_writelock(&lock);
return client;
}
}
printf("Client %i not found \n", client);
rwlock_release_writelock(&lock);
return 0;
}

void traverseList(int client){ //take client sock to send to?
rwlock_acquire_readlock(&lock);
char listOfConnections[BUF_SIZE];
myNode* current = linkedList;
memset(listOfConnections, 0, BUF_SIZE);
strcat(&listOfConnections, "Available Clients: \n");
while (current != NULL){
//printf("Name of client: %s\n", current->clientName);
/// put this in a variable, and send it to client?
char* name = current->clientName;
char str[] = "Client Identification: ";
int lineSize = strlen(name) + strlen(str) + 1;
char line[lineSize];
sprintf(line, "%s %s \n", str, name);
strcat(&listOfConnections, line);
current = current->next;
}
rwlock_release_readlock(&lock);
if ( write(client, listOfConnections, strlen(listOfConnections)) <
    0 ){
perror("send");
}

}


int checkValidityClient(char* name, int n){
rwlock_acquire_readlock(&lock);
myNode* current = linkedList;

while (current != NULL){
if (strcmp(current->clientName, name) == 0){
rwlock_release_readlock(&lock);
return -1;
}
current = current->next;
}
rwlock_release_readlock(&lock);
return 0;
}

int getIDfromName(char* name){
//given a client's id, get it's socket number in order to send
    message
int client2 = 0;
rwlock_acquire_readlock(&lock);
myNode* current = linkedList;
while (current != NULL && (strcmp(current->clientName, name)) != 0)
    {

current = current->next;
}
if (current != NULL && (strcmp(current->clientName, name)) == 0) {
client2 =  current->clientSock;
}
rwlock_release_readlock(&lock);
return client2;
}



void* clientThreadNew(void* clientNod){
myNode* clientNode = (myNode*) clientNod;
int n;
char response[BUF_SIZE];
enque(clientNode);
while (1){

memset(response, 0, BUF_SIZE);
if((n = read(clientNode->clientSock, response, BUF_SIZE-1)) < 0){
perror("read");
deque(clientNode->clientSock);
close(clientNode->clientSock);
pthread_exit(NULL);
return NULL;
}
```

```c
else{

response[n] = 0;
char msg[strlen(response)+1];
strcpy(msg, response);
msg[strlen(response)] = 0;
printf("%d sent a message/response: %s \n", clientNode->clientSock,
    msg);

//client sends a command, store it in msg
// msg = client ki command;
if (strcmp(msg, listCommand) == 0) {
traverseList(clientNode->clientSock); //call with client id to
    write to
continue;
}

if (strncmp(msg, msgCommand, 4) == 0) {
int i;
int firstSpace = 0; int secondSpace = 0;
int clientNameIdx = -1;
int msgIdx = -1;
for(i=4; i<BUF_SIZE; i++){ //starting after /msg
if (msg[i] == ' ')  // tokenize based on space.
{
if (secondSpace == 0 && clientNameIdx == -1) clientNameIdx = i+1;
if (firstSpace == 1 && secondSpace == 1 && msgIdx == -1) msgIdx = i
    +1;
}
if (msg[i] != ' '){
if (firstSpace == 0 && secondSpace == 0)
{
firstSpace = 1;
continue;
}
if (firstSpace == 1 && secondSpace == 0) secondSpace = 1;
}
}
if (clientNameIdx != -1){
while (msg[clientNameIdx] == ' ') clientNameIdx++;
}

i = 0;

char clientName[msgIdx - clientNameIdx];
int currIdx = 0;

for( i = clientNameIdx; i < msgIdx-1; i++){
clientName[currIdx] = msg[i];
currIdx++;
}

clientName[currIdx] = 0;
if (msgIdx != -1){
while (msg[msgIdx] == ' ') msgIdx++;
}

if (clientNameIdx == -1 || msgIdx == -1){
char sendMsg[BUF_SIZE];
strcpy(sendMsg, "Number of arguments not specified correctly,
    please enter /msg client_name message \n");
if ( write(clientNode->clientSock, sendMsg, strlen(sendMsg)) < 0 ){
perror("send");
}
continue;
}


char msgClient[strlen(msg) - clientNameIdx + 1];
i = 0;
currIdx = 0;

for( i = msgIdx; i < strlen(msg); i++){
msgClient[currIdx] = msg[i];
currIdx++;
}
char msgToClient[BUF_SIZE];
msgClient[currIdx] = 0;

int clientID = getIDfromName(clientName);
memset(msgToClient, 0, BUF_SIZE);
char str[] = "message from";
sprintf(msgToClient, "%s %s: %s \n", str, clientNode->clientName,
    msgClient);
if ( write(clientID, msgToClient, strlen(msgToClient)) < 0 ) {
perror("send");
}
continue;
//return 1; //command successfully execed.
}

if (strncmp(msg, quitCommand, strlen(quitCommand)) == 0) {
char sendMsg[BUF_SIZE];
strcpy(sendMsg, "Closing connection in 3....2.....1\n");
if ( write(clientNode->clientSock, sendMsg, strlen(sendMsg)) < 0 ){
perror("send");
}
close(clientNode->clientSock);
deque(clientNode->clientSock);
printf("Connection Closed: %d \n", clientNode->clientSock);
pthread_exit(NULL);
return 1; //command successfully execed.
}
else{
char sendMsg[BUF_SIZE];
strcpy(sendMsg, "Invalid command\n");
if ( write(clientNode->clientSock, sendMsg, strlen(sendMsg)) < 0 ){
perror("send");
}
}
}
}
}

```

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

if (argc < 2){
printf("Port? Please enter port, exiting......\n");
exit(1);
}

char hostname[]="127.0.0.1";   //localhost ip address to bind to
short port=atoi(argv[1]);                //the port we are to bind to

struct sockaddr_in saddr_in;  //socket interent address of server
struct sockaddr_in client_saddr_in;  //socket interent address of client

socklen_t saddr_len = sizeof(struct sockaddr_in); //length of address

int server_sock, client_sock;         //socket file descriptor


char response[BUF_SIZE];           //what to send to the client
int n;                             //length measure

//set up the address information
saddr_in.sin_family = AF_INET;
inet_aton(hostname, &saddr_in.sin_addr);
saddr_in.sin_port = htons(port);

//printf(" YAHAN MASLA NHE HAI\n");
//open a socket
if( (server_sock = socket(AF_INET, SOCK_STREAM, 0))  < 0){
perror("socket");
exit(1);
}

//bind the socket
if(bind(server_sock, (struct sockaddr *) &saddr_in, saddr_len) < 0){
perror("bind");
exit(1);
}

//ready to listen, queue up to 5 pending connectinos
if(listen(server_sock, 5)  < 0){
perror("listen");
exit(1);
}


saddr_len = sizeof(struct sockaddr_in); //length of address

printf("Listening On: %s:%d\n", inet_ntoa(saddr_in.sin_addr), ntohs(saddr_in.sin_port));

//accept incoming connections

```

```

while(1){
if((client_sock = accept(server_sock, (struct sockaddr *) &
    client_saddr_in, &saddr_len)) < 0){
//perror("accept");
//exit(1);
printf("%s\n", "connection failed\n");
}


//read from client
if((n = read(client_sock, response, BUF_SIZE-1)) < 0){
perror("read");
close(client_sock);
exit(1);
}

response[n] = '\0'; //NULL terminate string
//printf(" YAHAN MASLA NHE HAI 2\n");
myNode* clientNode = (myNode*) (malloc(sizeof(myNode)));
clientNode->clientSock = client_sock;
clientNode->clientName = (char*) (malloc(strlen(response) + 1));
strcpy(clientNode->clientName, response);
clientNode->clientName[strlen(response)] = '\0';
int valid = 0;
//printf("%s,response\n", response);
//printf("%s,clientName\n", clientNode->clientName);
if ( ( valid = checkValidityClient(response, strlen(response)) ) <
    0 ){
memset(response, 0, strlen(response));
strcpy(response, "Name clashes with another client, sorryyyyyyy\n")
    ;
if ( write(client_sock, response, strlen(response)) < 0 ){
perror("send");
}
free(clientNode->clientName);
free(clientNode);
close(client_sock);
}
if (valid == 0){

printf("Incoming Connection From: %s\n", clientNode->clientName);

strcpy(response, "establishing connection with server....\n");
if ( write(client_sock, response, strlen(response)) < 0 ) {
perror("send");
free(clientNode->clientName);
free(clientNode);
printf("Closing client socket: %i \n", client_sock);
close(client_sock);
}

int thread = 0;

if ( (thread = pthread_create(&clientThread, NULL, clientThreadNew,
     (void *)clientNode)) < 0 ){
printf("Failed to create connection, sorryyyyyyy\n");
free(clientNode->clientName);
free(clientNode);
}
}
}

printf("Closing socket\n");
close(server_sock);

return 0; //success
}
```

3 Makefile

```
all:
  gcc -o server -Wall gp18_server.c -lpthread
  gcc -o client -Wall gp18_client.c -lpthread

clear:
  rm client
  rm server
```

4 Comments

4.1 Help Taken

We took help from the course book for the code on locks. Also discussed the assignment with Rayyan.

4.2 Discrepancies

No discrepancies far as we know.

4.3 Comments about assignment

Good assignment, please give full marks.