

# CS232 Operating Systems

## Assignment 04: Introduction to Socket Programming

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

Fall 2019

### 1 Client

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <pthread.h>
6 #include <netinet/in.h>
7 #include <arpa/inet.h>
8 #include <string.h>
9 #include <netdb.h>
10
11 #define BUF_SIZE 4096
12
13 void * threadedInput(void * socket)
14 {
15     int sock = *((int *) socket);
16     char *inputBuffer;
17     int inputBufferSize = 120;
18
19     inputBuffer = (char *)malloc(inputBufferSize * sizeof(char));
20
21     while(1){
22         if (getline(&inputBuffer, (size_t*) &inputBufferSize, stdin) != 0)
23         {
24             //printf("%s\n", inputBuffer);
25             if(write(sock, inputBuffer, strlen(inputBuffer)) < 0) perror("send");
26         }
27         free(inputBuffer);
28         return NULL;
29     }
30
31     int main(int argc, char* argv[]){
32         if (argc != 4){
33             printf("%s\n", "please pass appropriate arguments... exiting");
34             exit(1);
```

```

35 }
36
37 char * hostname = argv[1]; //the hostname we are looking up
38 short port = atoi(argv[2]); //the port we are
    connecting on
39
40 struct addrinfo *result; //to store results
41 struct addrinfo hints; //to indicate information we want
42
43 struct sockaddr_in *saddr_in; //socket internet address
44
45 int s, n; //for error checking
46
47 int client_socket; //socket file descriptor
48
49 pthread_t client_thread;
50
51 char* client_name = argv[3];
52
53 char response[BUF_SIZE]; //read in 4096 byte chunks
54
55 //setup our hints
56 memset(&hints, 0, sizeof(struct addrinfo)); //zero out hints
57 hints.ai_family = AF_INET; //we only want IPv4 addresses
58
59 //Convert the hostname to an address
60 if( (s = getaddrinfo(hostname, NULL, &hints, &result)) != 0){
61     fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
62     exit(1);
63 }
64
65 //convert generic socket address to inet socket address
66 saddr_in = (struct sockaddr *) result->ai_addr;
67
68 //set the port in network byte order
69 saddr_in->sin_port = htons(port);
70
71 //open a socket
72 if( (client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0){
73     perror("socket");
74     exit(1);
75 }
76
77 //connect to the server
78 if( connect(client_socket, (struct sockaddr *) saddr_in, sizeof(*
    saddr_in)) < 0 ) {
79     perror("connect");
80     exit(1);
81 }
82
83 //send the client_name
84 if( write(client_socket, client_name, strlen(client_name)) < 0 ){
85     perror("send");
86 }
87
88 //read the response until EOF
89 pthread_create(&client_thread, NULL, threadedInput, &client_socket)

```

```

90     ;
91     while (1)
92     {
93         memset(response, 0, BUF_SIZE);
94         n = read(client_socket, response, BUF_SIZE-1);
95         if(n <= 0){ //close the socket
96             close(client_socket);
97             printf("Socket Closed %d\n:", client_socket);
98             exit(1);
99             return 0;
100         }
101         else {
102             printf("%s", response);
103         }
104     }
105     return 0; //success
106 }

```

## 2 Server

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  #include <pthread.h>
6  #include <netinet/in.h>
7  #include <arpa/inet.h>
8  #include <string.h>
9  #include <netdb.h>
10 #include <semaphore.h>
11
12
13 #define BUF_SIZE 4096
14
15 char* listCommand = "/list\n";
16
17 char* msgCommand = "/msg";
18
19 char* quitCommand = "/quit";
20
21
22
23 pthread_t clientThread;
24
25 typedef struct __node {
26     char* clientName;
27     int clientSock; //file descriptor
28     struct __node* next;
29 } myNode;
30
31 myNode* linkedList = NULL; //head
32
33 typedef struct _rwlock_t {
34     sem_t lock; // binary semaphore (basic lock)
35     sem_t writelock; // allow ONE writer/MANY readers
36     int readers; // #readers in critical section

```

```

37 } rwlock_t;
38
39 rwlock_t lock;
40
41 void rwlock_init(rwlock_t *rw) {
42     rw->readers = 0;
43     sem_init(&rw->lock, 0, 1);
44     sem_init(&rw->writelock, 0, 1);
45 }
46
47 void rwlock_acquire_readlock(rwlock_t *rw) {
48     sem_wait(&rw->lock);
49     rw->readers++;
50     if (rw->readers == 1) // first reader gets writelock
51         sem_wait(&rw->writelock);
52     sem_post(&rw->lock);
53 }
54
55 void rwlock_release_readlock(rwlock_t *rw) {
56     sem_wait(&rw->lock);
57     rw->readers--;
58     if (rw->readers == 0) // last reader lets it go
59         sem_post(&rw->writelock);
60     sem_post(&rw->lock);
61 }
62
63 void rwlock_acquire_writelock(rwlock_t *rw) {
64     sem_wait(&rw->writelock);
65 }
66
67 void rwlock_release_writelock(rwlock_t *rw) {
68     sem_post(&rw->writelock);
69 }
70
71
72 void enqueue(myNode* new){
73     rwlock_acquire_writelock(&lock);
74     myNode* temp = linkedList;
75     if (linkedList == NULL){
76         linkedList = new;
77         linkedList->next = NULL;
78     }
79     rwlock_release_writelock(&lock);
80     return;
81 }
82 else {
83     new->next = linkedList;
84     new->next->next = temp->next;
85 }
86 linkedList = new;
87 rwlock_release_writelock(&lock);
88 }
89
90 int deque(int client){ //remove client having file descriptor
91     client
92     rwlock_acquire_writelock(&lock);
93     myNode* current = linkedList;
94     myNode* previous = linkedList;

```

```

93 //int flag = 0;
94
95
96 if (linkedList == NULL) { //if linked list is empty, release lock
    and return
97 rwlock_release_writelock(&lock);
98 return 0;
99 }
100
101 if (linkedList->clientSock == client){
102     linkedList = linkedList->next;
103     printf("Client %i Removed....\n", client);
104     free(current->clientName);
105     free(current);
106     rwlock_release_writelock(&lock);
107     return client; //just to check if the client that we removed is
        infact the client that we wanted to remove
108 }
109
110 while (current->next != NULL && current->clientSock != client){
111     previous = current;
112     current = current->next;
113 }
114 if (current != NULL){
115     if (current->clientSock == client){
116         printf("Client %i Removed....\n", client);
117         previous->next = current->next;
118         free(current->clientName);
119         free(current);
120         rwlock_release_writelock(&lock);
121         return client;
122     }
123 }
124 printf("Client %i not found \n", client);
125 rwlock_release_writelock(&lock);
126 return 0;
127 }
128
129 void traverseList(int client){ //take client sock to send to?
130     rwlock_acquire_readlock(&lock);
131     char listOfConnections[BUF_SIZE];
132     myNode* current = linkedList;
133     memset(listOfConnections, 0, BUF_SIZE);
134     strcat(&listOfConnections, "Available Clients: \n");
135     while (current != NULL){
136         //printf("Name of client: %s\n", current->clientName);
137         /// put this in a variable, and send it to client?
138         char* name = current->clientName;
139         char str[] = "Client Identification: ";
140         int lineSize = strlen(name) + strlen(str) + 1;
141         char line[lineSize];
142         sprintf(line, "%s %s \n", str, name);
143         strcat(&listOfConnections, line);
144         current = current->next;
145     }
146     rwlock_release_readlock(&lock);
147     if ( write(client, listOfConnections, strlen(listOfConnections)) <

```

```

    0 ){
148 perror("send");
149 }
150
151 }
152
153
154 int checkValidityClient(char* name, int n){
155     rwlock_acquire_readlock(&lock);
156     myNode* current = linkedList;
157
158     while (current != NULL){
159         if (strcmp(current->clientName, name) == 0){
160             rwlock_release_readlock(&lock);
161             return -1;
162         }
163         current = current->next;
164     }
165     rwlock_release_readlock(&lock);
166     return 0;
167 }
168
169 int getIDfromName(char* name){
170     //given a client's id, get it's socket number in order to send
171     //message
172     int client2 = 0;
173     rwlock_acquire_readlock(&lock);
174     myNode* current = linkedList;
175     while (current != NULL && (strcmp(current->clientName, name)) != 0)
176     {
177         current = current->next;
178     }
179     if (current != NULL && (strcmp(current->clientName, name)) == 0) {
180         client2 = current->clientSock;
181     }
182     rwlock_release_readlock(&lock);
183     return client2;
184 }
185
186
187 void* clientThreadNew(void* clientNod){
188     myNode* clientNode = (myNode*) clientNod;
189     int n;
190     char response[BUF_SIZE];
191     enqueue(clientNode);
192     while (1){
193         memset(response, 0, BUF_SIZE);
194         if ((n = read(clientNode->clientSock, response, BUF_SIZE-1)) < 0){
195             perror("read");
196             deque(clientNode->clientSock);
197             close(clientNode->clientSock);
198             pthread_exit(NULL);
199         }
200         return NULL;
201     }

```

```

202 else{
203
204 response[n] = 0;
205 char msg[strlen(response)+1];
206 strcpy(msg, response);
207 msg[strlen(response)] = 0;
208 printf("%d sent a message/response: %s \n", clientNode->clientSock,
      msg);
209
210 //client sends a command, store it in msg
211 // msg = client ki command;
212 if (strcmp(msg, listCommand) == 0) {
213 traverseList(clientNode->clientSock); //call with client id to
      write to
214 continue;
215 }
216
217 if (strncmp(msg, msgCommand, 4) == 0) {
218 int i;
219 int firstSpace = 0; int secondSpace = 0;
220 int clientNameIdx = -1;
221 int msgIdx = -1;
222 for(i=4; i<BUF_SIZE; i++){ //starting after /msg
223 if (msg[i] == ' ') // tokenize based on space.
224 {
225 if (secondSpace == 0 && clientNameIdx == -1) clientNameIdx = i+1;
226 if (firstSpace == 1 && secondSpace == 1 && msgIdx == -1) msgIdx = i
      +1;
227 }
228 if (msg[i] != ' '){
229 if (firstSpace == 0 && secondSpace == 0)
230 {
231 firstSpace = 1;
232 continue;
233 }
234 if (firstSpace == 1 && secondSpace == 0) secondSpace = 1;
235 }
236 }
237 if (clientNameIdx != -1){
238 while (msg[clientNameIdx] == ' ') clientNameIdx++;
239 }
240
241 i = 0;
242
243 char clientName[msgIdx - clientNameIdx];
244 int currIdx = 0;
245
246 for( i = clientNameIdx; i < msgIdx-1; i++){
247 clientName[currIdx] = msg[i];
248 currIdx++;
249 }
250
251 clientName[currIdx] = 0;
252 if (msgIdx != -1){
253 while (msg[msgIdx] == ' ') msgIdx++;
254 }
255

```

```

256 if (clientNameIdx == -1 || msgIdx == -1){
257     char sendMsg[BUF_SIZE];
258     strcpy(sendMsg, "Number of arguments not specified correctly ,
        please enter /msg client_name message \n");
259     if ( write(clientNode->clientSock , sendMsg, strlen(sendMsg)) < 0 ){
260         perror("send");
261     }
262     continue;
263 }
264
265
266 char msgClient[ strlen(msg) - clientNameIdx + 1];
267 i = 0;
268 currIdx = 0;
269
270 for( i = msgIdx; i < strlen(msg); i++){
271     msgClient[currIdx] = msg[i];
272     currIdx++;
273 }
274 char msgToClient[BUF_SIZE];
275 msgClient[currIdx] = 0;
276
277 int clientID = getIDfromName(clientName);
278 memset(msgToClient, 0, BUF_SIZE);
279 char str[] = "message from";
280 sprintf(msgToClient, "%s %s: %s \n", str, clientNode->clientName,
        msgClient);
281 if ( write(clientID , msgToClient, strlen(msgToClient)) < 0 ) {
282     perror("send");
283 }
284 continue;
285 //return 1; //command successfully excecud.
286 }
287
288 if (strncmp(msg, quitCommand, strlen(quitCommand)) == 0) {
289     char sendMsg[BUF_SIZE];
290     strcpy(sendMsg, "Closing connection in 3....2.....1\n");
291     if ( write(clientNode->clientSock , sendMsg, strlen(sendMsg)) < 0 ){
292         perror("send");
293     }
294     close(clientNode->clientSock);
295     deque(clientNode->clientSock);
296     printf("Connection Closed: %d \n", clientNode->clientSock);
297     pthread_exit(NULL);
298     return 1; //command successfully excecud.
299 }
300 else{
301     char sendMsg[BUF_SIZE];
302     strcpy(sendMsg, "Invalid command\n");
303     if ( write(clientNode->clientSock , sendMsg, strlen(sendMsg)) < 0 ){
304         perror("send");
305     }
306 }
307 }
308 }
309 }
310

```



```

311 int main(int argc, char* argv[]) {
312
313     if (argc < 2){
314         printf("Port? Please enter port, exiting.....\n");
315         exit(1);
316     }
317
318     char hostname[]="127.0.0.1";    //localhost ip address to bind to
319     short port=atoi(argv[1]);      //the port we are to bind
        to
320
321     struct sockaddr_in saddr_in;    //socket interent address of server
322     struct sockaddr_in client_saddr_in; //socket interent address of
        client
323
324     socklen_t saddr_len = sizeof(struct sockaddr_in); //length of
        address
325
326     int server_sock, client_sock;    //socket file descriptor
327
328
329     char response[BUF_SIZE];         //what to send to the client
330     int n;                           //length measure
331
332     //set up the address information
333     saddr_in.sin_family = AF_INET;
334     inet_aton(hostname, &saddr_in.sin_addr);
335     saddr_in.sin_port = htons(port);
336
337     //printf(" YAHAN MASLA NHE HAI\n");
338     //open a socket
339     if( (server_sock = socket(AF_INET, SOCK_STREAM, 0)) < 0){
340         perror("socket");
341         exit(1);
342     }
343
344     //bind the socket
345     if(bind(server_sock, (struct sockaddr *) &saddr_in, saddr_len) < 0)
        {
346         perror("bind");
347         exit(1);
348     }
349
350     //ready to listen, queue up to 5 pending connectinos
351     if(listen(server_sock, 5) < 0){
352         perror("listen");
353         exit(1);
354     }
355
356
357     saddr_len = sizeof(struct sockaddr_in); //length of address
358
359     printf("Listening On: %s:%d\n", inet_ntoa(saddr_in.sin_addr), ntohs
        (saddr_in.sin_port));
360
361     //accept incoming connections
362

```

```

363 while(1){
364     if((client_sock = accept(server_sock, (struct sockaddr *) &
365         client_saddr_in, &saddr_len)) < 0){
366         perror("accept");
367         exit(1);
368         printf("%s\n", "connection failed\n");
369     }
370
371     //read from client
372     if((n = read(client_sock, response, BUF.SIZE-1)) < 0){
373         perror("read");
374         close(client_sock);
375         exit(1);
376     }
377
378     response[n] = '\0'; //NULL terminate string
379     //printf(" YAHAN MASLA NHE HAI 2\n");
380     myNode* clientNode = (myNode*) (malloc(sizeof(myNode)));
381     clientNode->clientSock = client_sock;
382     clientNode->clientName = (char*) (malloc(strlen(response) + 1));
383     strcpy(clientNode->clientName, response);
384     clientNode->clientName[strlen(response)] = '\0';
385     int valid = 0;
386     //printf("%s,response\n", response);
387     //printf("%s,clientName\n", clientNode->clientName);
388     if ( ( valid = checkValidityClient(response, strlen(response)) ) <
389         0 ){
390         memset(response, 0, strlen(response));
391         strcpy(response, "Name clashes with another client, sorryyyyyyy\n");
392         if ( write(client_sock, response, strlen(response)) < 0 ){
393             perror("send");
394         }
395         free(clientNode->clientName);
396         free(clientNode);
397         close(client_sock);
398     }
399     if (valid == 0){
400
401         printf("Incoming Connection From: %s\n", clientNode->clientName);
402
403         strcpy(response, "establishing connection with server...\n");
404         if ( write(client_sock, response, strlen(response)) < 0 ) {
405             perror("send");
406             free(clientNode->clientName);
407             free(clientNode);
408             printf("Closing client socket: %i \n", client_sock);
409             close(client_sock);
410         }
411
412         int thread = 0;
413
414         if ( (thread = pthread_create(&clientThread, NULL, clientThreadNew,
415             (void *)clientNode)) < 0 ){
416             printf("Failed to create connection, sorryyyyyyy\n");

```

```

416 free(clientNode->clientName);
417 free(clientNode);
418 }
419 }
420 }
421
422 printf("Closing socket\n");
423 close(server_sock);
424
425 return 0; //success
426 }

```

### 3 Makefile

```

1 all:
2     gcc -o server -Wall gp18_server.c -lpthread
3     gcc -o client -Wall gp18_client.c -lpthread
4
5 clear:
6     rm client
7     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.