Advanced Computer Communications

Assignment

(CNCO3002)

10th October 2022

Family Name : Fernando

Other Names : Naveen Pabodha

CURTIN ID : 20897525

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Fernando	Student ID:	20897525
Other name(s):	Naveen Pabodha		
Unit name:	Advanced Computer Communications	Unit ID:	CNCO3002
Lecturer / unit coordinator:	Dr. Sie Teng Soh	Tutor:	Mr. U.U. Samantha
Date of submission:	10/10/2022	Which assignment?	Assignm (Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: Fernando Date of signature: 10/10/2022

(By submitting this form, you indicate that you agree with all the above text.)

Contents

Source Code

Make File “makeclient”

```
client_snc.out: error.o wrapunix.o wrapsock.o wraplib.o wrapstdio.o client_snc.o readline.o writen.o wrappthread.o strclithread.o
	cc -o client_snc.out wrappthread.o strclithread.o error.o wrapunix.o wrapsock.o wraplib.o wrapstdio.o client_snc.o readline.o writen.o -lpthread

error.o: error.c acc.h
	cc -c error.c

wrapunix.o: wrapunix.c acc.h
	cc -c wrapunix.c

wrapsock.o: wrapsock.c acc.h
	cc -c wrapsock.c

wraplib.o: wraplib.c acc.h
	cc -c wraplib.c

wrapstdio.o: wrapstdio.c acc.h
	cc -c wrapstdio.c

client_snc.o: client_snc.c acc.h
	cc -c client_snc.c

strclithread.o: strclithread.c acc.h
	cc -c strclithread.c

readline.o: readline.c acc.h
	cc -c readline.c

writen.o: writen.c acc.h
	cc -c writen.c
wrappthread.o: wrappthread.c acc.h
	cc -c wrappthread.c

```

This file will run “.c” files that need to be compiled for client_snc.c code. Make file will use “-lpthread” because server and client programs use thread when a client is connected with the Server. After compiling the make file using “make -f makeclient” it will create “.out” for a run.

Make File “makeserver”

```
server_snc.out: error.o wrapunix.o wrapsock.o wraplib.o wrapstdio.o server_snc.o readline.o writen.o wrappthread.o
	cc -o server_snc.out wrappthread.o error.o wrapunix.o wrapsock.o wraplib.o wrapstdio.o server_snc.o readline.o writen.o -lpthread

error.o: error.c acc.h
	cc -c error.c

wrapunix.o: wrapunix.c acc.h
	cc -c wrapunix.c

wrapsock.o: wrapsock.c acc.h
	cc -c wrapsock.c

wraplib.o: wraplib.c acc.h
	cc -c wraplib.c

wrapstdio.o: wrapstdio.c acc.h
	cc -c wrapstdio.c

server_snc.o: server_snc.c acc.h
	cc -c server_snc.c

readline.o: readline.c acc.h
	cc -c readline.c

writen.o: writen.c acc.h
	cc -c writen.c
wrappthread.o: wrappthread.c acc.h
	cc -c wrappthread.c
```

This file will run “.c” files that need to be compiled for server_snc.c code. Make file will use “-lpthread” because server and client programs use thread when a client is connected with the Server. After compiling the make file using “make -f makeserver” it will create “.out” for a run.

Code file “acc. h”

```c
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <string.h>
#include <errno.h>
#include <netdb.h>
#include <sys/uio.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <pthread.h>
#include <stdio.h>/*scanf*/
#include <signal.h>
#include <fcntl.h>
#include <stdbool.h>

#ifdef  HAVE_PTHREAD_H
#include    <pthread.h>
#endif

static int  max_idle_time; //Maximum time the server waits

#define MAXLINE 4096
#define LISTENQ 1024
#define SERV_TCP_PORT   52002
#define OK  52001
#define SA  struct sockaddr
#define  min(a,b) ((a) < (b) ? (a) : (b))
#define  max(a,b) ((a) > (b) ? (a) : (b))
/* Our own header for the programs that use threads.
   Include this file, instead of "unp.h". */

typedef  void  Sigfunc(int);

#ifndef     __unp_pthread_h
#define     __unp_pthread_h

void    Pthread_create(pthread_t *, const pthread_attr_t *, void * (*)(void *), void *);
   /*void * (*)(void *), void *);*/
void    Pthread_join(pthread_t, void **);
void    Pthread_detach(pthread_t);
void    Pthread_kill(pthread_t, int);

void    Pthread_mutexattr_init(pthread_mutexattr_t *);
void    Pthread_mutexattr_setpshared(pthread_mutexattr_t *, int);
void    Pthread_mutex_init(pthread_mutex_t *, pthread_mutexattr_t *);
void    Pthread_mutex_lock(pthread_mutex_t *);
void    Pthread_mutex_unlock(pthread_mutex_t *);

void    Pthread_cond_broadcast(pthread_cond_t *);
void    thread_cond_signal(pthread_cond_t *);
void    Pthread_cond_wait(pthread_cond_t *, pthread_mutex_t *);
```

```
56 void    Pthread_cond_timedwait(pthread_cond_t *, pthread_mutex_t *,
57                                  const struct timespec *);
58
59 void    Pthread_key_create(pthread_key_t *, void (*)(void *));
60 void    Pthread_setspecific(pthread_key_t, const void *);
61 void    Pthread_once(pthread_once_t *, void (*)(void));
62
63 #endif  /* __unp_pthread_h */
```

This file includes all header files that we use in source code and definitions values that use for indicating MAXLINE, LISTENNQ, SERV_TCP_PORT, etc. Also, a global variable that operates in both the Server and client.

Code file “client_snc. c”

```c
#include    "acc.h"

void    *copyto(void *);

static int  sockfd;     /* global for both threads to access */
static FILE *fp;

int
main(int argc, char **argv){

    int         sockfd,res,arg;
    struct sockaddr_in  servaddr;
    fd_set myset;
    struct timeval tv;
    int valopt;
    socklen_t lon;

    char            *ptr, **pptr;
    char            str[INET_ADDRSTRLEN];
    struct hostent  *hptr;

    //check argument are correctly entered by user
    if (argc != 3)
        err_quit("usage: tcpcli <IPaddress> <port number>");

    sockfd = Socket(AF_INET, SOCK_STREAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(atoi(argv[2]));
    Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

    /*
    //set non-blocking
    if((arg = fcntl(sockfd, F_GETFL, NULL)) < 0){
        fprintf(stderr, "Error fcntl( ... , F_GETFL) (%s)\n", strerror(errno));
    }
    arg |= SOCK_NONBLOCK;
    if(fcntl(sockfd, F_SETFL, arg) < 0){
        fprintf(stderr, "Error fcntl( ... , F_SETFL) (%s)\n", strerror(errno));
        exit(0);
    }
    */

    if ( (hptr = gethostbyname2(*++argv, AF_INET)) == NULL) {
            err_msg("gethostbyname error for host: %s: %s",*++argv, hstrerror(h_errno));
    }

    printf(" ____________________________________________________________________________\n");
    printf("|                \n");
    printf("|official hostname: %s\n", hptr->h_name);
    printf("|address type: %d\n", hptr->h_addrtype);
    printf("|address length: %d\n", hptr->h_length);
    printf("|____________________________________________________________________________\n");

    res = Connect(sockfd, (SA *) &servaddr, sizeof(servaddr));//connecting with server

//(not working - idle_time handling another methode but running errors)_____________________________
/*
    if(res < 0){
        if(errno == EINPROGRESS){
            fprintf(stderr, "EINPROGRESS in connect() - selecting\n");
            do{
                tv.tv_sec = max_idle_time;
                tv.tv_usec = 0;
                FD_ZERO(&myset);
                FD_SET(sockfd,&myset);
                res = select(sockfd+1, NULL, &myset, NULL, &tv);
```

```
                if(res < 0 && errno != EINTR){
                    fprintf(stderr, "Error connecting %d - %s\n", errno, strerror(errno));
                    exit(0);
                }else if(res > 0){
                    //socket selected for write
                    lon = sizeof(int);
                    if(getsockopt(sockfd, SOL_SOCKET, SO_ERROR, (void*)(&valopt), &lon) < 0){
                        fprintf(stderr, "Error in getsockopt() %d - %s\n", errno, strerror(errno));
                        exit(0);
                    }
                    //check the value returned
                    if(valopt){
                        fprintf(stderr, "Error in delayed connection() %d -%s\n",valopt,strerror(valopt));
                        exit(0);
                    }
                    break;
                }else {
                    fprintf(stderr, "Timeout in select() - Cancelling!\n");
                    exit(0);
                }
            }while(1);
        }else{
            fprintf(stderr, "Error connecting %d - %s\n", errno,strerror(errno));
            exit(0);
        }
    }
    //set to blocking mode again
    if((arg = fcntl(sockfd, F_GETFL,NULL)) < 0){
        fprintf(stderr,"Error fcntl( ... , F_GETFL) (%s)\n", strerror(errno));
        exit(0);
    }
    arg &= (~SOCK_NONBLOCK);
    if(fcntl(sockfd, F_SETFL,arg) < 0){
        fprintf(stderr, "Error fcntl( ... , F_SETFL) (%s)\n", strerror(errno));
        exit(0);
    }
*/
//(welcome message and str_cli() calling)-------------------------------------------------------------------------
    ------------------------

    printf(" _____________________________________________________________________________\n");
    printf("|                        \n");
    printf("|                                Welcome to SNC!  \n");
    printf("|                                    Client  \n");
    printf("|_____________________________________________________________________________\n\n");

    str_cli(stdin, sockfd);     /* do it all */

    exit(0);
}
```

In “client_snc.c” file is the one creating the client communication environment for the user. This file is included in “server_snc.out” file and will be the main code run first. This will execute and get user arguments to create a socket for communication (IP address, port number), and gethostbyname2() will get client data and print it in the client console. Then connect() will start three-way handshake with the Server. After successfully establishing the connection, it will print a welcome message and “Str_cli()” function will be called for further code processing.

Code file “strclithread. c”

```c
#include    "acc.h"

void    *copyto(void *);

static int  sockfd;     /* global for both threads to access */
static FILE *fp;

pthread_mutex_t   counter_mutex = PTHREAD_MUTEX_INITIALIZER;

//stdout print Student ID 20897525 (working)
void overwrite_stdout() {
    printf("\r%s", "20897525 > ");
    fflush(stdout);
}

void
str_cli(FILE *fp_arg, int sockfd_arg)
{
    char        recvline[MAXLINE];
    pthread_t   tid;

    sockfd = sockfd_arg;    /* copy arguments to externals */
    fp = fp_arg;

    printf("20897525 > ");//fist Id output(1 time)

    /* does no pass argument */
    Pthread_create(&tid, NULL, copyto, NULL);
    //Pthread_create(&tid, NULL, idle_time, NULL);

    while (Readline(sockfd, recvline, MAXLINE) > 0){
        pthread_mutex_lock(&counter_mutex);

        Fputs(recvline, stdout);
        overwrite_stdout();//print ID>(working)

        pthread_mutex_unlock(&counter_mutex);
    }
}

void *
copyto(void *arg)
{
    char    sendline[MAXLINE];

    while ((void *) Fgets(sendline, MAXLINE, fp) != NULL){
        pthread_mutex_lock(&counter_mutex);

        //to quit from client(working)
        if (strncasecmp(sendline, "QUIT",4) == 0){
            sleep(1);//sleep process for run other code without sudden forcefull termination
            close(sockfd);
            exit(1);
        }

        Writen(sockfd, sendline, strlen(sendline));

        pthread_mutex_unlock(&counter_mutex);
    }

    Shutdown(sockfd, SHUT_WR);  /* EOF on stdin, send FIN */

    return(NULL);
    /* 4return (i.e., thread terminates) when end-of-file on stdin */
}
```

“Strclithread.c” is the one that include “str_cli ()” function. This function will first copy the argument for created variable inside the function and print “20897525>” for one time, and it will create a thread for a client (every client will have a separate thread) that is connected with “copyto ()” function. Then the while loop will read data that type in the client console and move that data to stdout port for send. Then “overwrite_stdout ()” will add “20897525>” id for every line on the console before allowing the user to type.

“copyto ()” function will get sending data and write in socket for sending to the Server. And if the condition reads that the user type “quit”, then the program will close(sockfd) and exit from the chat room.

In both loops mutex lock and mutex unlock are used to prevent scrambling client data with other client data.

Code file “server_snc. c”

```c
#include    "acc.h"

static void *doit(void *);      /* each thread executes this function */
static _Atomic unsigned int cli_count = 0;
//static int max_idle_time ; //Maximum time the server waits
static int m ; //Maximum number of Clients

static char connfd_c[10][100], nickname[10][100],realname[10][100],buffer[256];//array creation and buffer for string storing

pthread_mutex_t counter_mutex = PTHREAD_MUTEX_INITIALIZER;

//------------------------------------------------------------------------------
union val {
  int               i_val;
  long              l_val;
  char              c_val[10];
  struct linger     linger_val;
  struct timeval    timeval_val;
} val;

static char *sock_str_flag(union val *, int);
static char *sock_str_int(union val *, int);
static char *sock_str_linger(union val *, int);
static char *sock_str_timeval(union val *, int);

struct sock_opts {
  char     *opt_str;
  int       opt_level;
  int       opt_name;
  char   *(*opt_val_str)(union val *, int);
} sock_opts[] = {
/*
//(get sock details - removing comment can run this part)-------------------------------------------------------------------------------
    "SO_BROADCAST",     SOL_SOCKET, SO_BROADCAST,   sock_str_flag,
    "SO_DEBUG",         SOL_SOCKET, SO_DEBUG,       sock_str_flag,
    "SO_DONTROUTE",     SOL_SOCKET, SO_DONTROUTE,   sock_str_flag,
    "SO_ERROR",         SOL_SOCKET, SO_ERROR,       sock_str_int,
    "SO_KEEPALIVE",     SOL_SOCKET, SO_KEEPALIVE,   sock_str_flag,
    "SO_LINGER",        SOL_SOCKET, SO_LINGER,      sock_str_linger,
    "SO_OOBINLINE",     SOL_SOCKET, SO_OOBINLINE,   sock_str_flag,
    "SO_RCVBUF",        SOL_SOCKET, SO_RCVBUF,      sock_str_int,
    "SO_SNDBUF",        SOL_SOCKET, SO_SNDBUF,      sock_str_int,
    "SO_RCVLOWAT",      SOL_SOCKET, SO_RCVLOWAT,    sock_str_int,
    "SO_SNDLOWAT",      SOL_SOCKET, SO_SNDLOWAT,    sock_str_int,
    "SO_RCVTIMEO",      SOL_SOCKET, SO_RCVTIMEO,    sock_str_timeval,
    "SO_SNDTIMEO",      SOL_SOCKET, SO_SNDTIMEO,    sock_str_timeval,
    "SO_REUSEADDR",     SOL_SOCKET, SO_REUSEADDR,   sock_str_flag,
#ifdef  SO_REUSEPORT
    "SO_REUSEPORT",     SOL_SOCKET, SO_REUSEPORT,   sock_str_flag,
#else
    "SO_REUSEPORT",     0,          0,              NULL,
#endif
    "SO_TYPE",          SOL_SOCKET, SO_TYPE,        sock_str_int,
#ifdef  SO_USELOOPBACK
    "SO_USELOOPBACK",   SOL_SOCKET, SO_USELOOPBACK, sock_str_flag,
#else
    "SO_USELOOPBACK",       0,          0,              NULL,
#endif
    "IP_TOS",           IPPROTO_IP, IP_TOS,         sock_str_int,
    "IP_TTL",           IPPROTO_IP, IP_TTL,         sock_str_int,
    "TCP_MAXSEG",       IPPROTO_TCP,TCP_MAXSEG,     sock_str_int,
    "TCP_NODELAY",      IPPROTO_TCP,TCP_NODELAY,    sock_str_flag,
    NULL,               0,          0,              NULL
*/};
```

```c
// *INDENT-ON*

int* start_timer(void *secs);

//(messag sending and queue handling )-----------------------------------------------------------------

//Remove clients to queue (function is not used)
void queue_remove(int cli_count,char *nickname){
    pthread_mutex_lock(&counter_mutex);
    //null values that belog to quited client
    for(int i=0; i<m; ++i){
        if(strcasecmp(nickname[i], nickname[cli_count-1])){
            strcpy(nickname[i],NULL);
            strcpy(realname[i],NULL);
            strcpy(connfd_c[i],NULL);
        }
    }

    pthread_mutex_unlock(&counter_mutex);
}

//Send message to all clients except sender(code or structure error)
void send_message_all(char *nickname, int cli_count, char conn){
    pthread_mutex_lock(&counter_mutex);

    for(int i=0; i<m; ++i){
        if(strcasecmp(nickname[i], nickname[cli_count-1])){
            if(strcmp(connfd_c[i],conn) != 0){
                bzero(buffer,sizeof(buffer));
                sprintf(buffer, "Server : hi 3!\n");
                write(conn, buffer, sizeof(buffer));
                if(write(connfd_c[i], nickname, strlen(nickname)) < 0){
                    perror("ERROR: write to descriptor failed");
                    break;
                }
            }
        }
    }

    pthread_mutex_unlock(&counter_mutex);
}

//Send message to client using nickname(code or structure error)
void send_message_client(char *nickname, int cli_count, char *msg){
    pthread_mutex_lock(&counter_mutex);

    int conn;
    for(int i=0; i<m; ++i){
        //cheak nickname
        if(strcasecmp(nickname[i], nickname)){
            //conn need to replce to nickname client connfd
            if(strcmp(connfd_c[i],conn) == 0){
                bzero(buffer,sizeof(buffer));
                sprintf(buffer, "%s : %s\n",nickname, msg);//message save in buffer
                write(conn, buffer, sizeof(buffer));//write buffer to client
                if(write(connfd_c[i], nickname, strlen(nickname)) < 0)
                {
                    perror("ERROR: write to descriptor failed");
                    break;
                }
            }
        }
    }

    pthread_mutex_unlock(&counter_mutex);
}
```

```c
//(timeout function-not tested/not working properly)-------------------------------------------------------------------------
static char strres[128];

static char *
sock_str_flag(union val *ptr, int len){
/* *INDENT-OFF* */
    if (len != sizeof(int))
        snprintf(strres, sizeof(strres), "size (%d) not sizeof(int)", len);
    else
        snprintf(strres, sizeof(strres),
                 "%s", (ptr->i_val == 0) ? "off" : "on");
    return(strres);
/* *INDENT-ON* */
}
/* end checkopts3 */

static char *
sock_str_int(union val *ptr, int len){
    if (len != sizeof(int))
        snprintf(strres, sizeof(strres), "size (%d) not sizeof(int)", len);
    else
        snprintf(strres, sizeof(strres), "%d", ptr->i_val);
    return(strres);
}

static char *
sock_str_linger(union val *ptr, int len){
    struct linger   *lptr = &ptr->linger_val;

    if (len != sizeof(struct linger))
        snprintf(strres, sizeof(strres),
                 "size (%d) not sizeof(struct linger)", len);
    else
        snprintf(strres, sizeof(strres), "l_onoff = %d, l_linger = %d",
                 lptr->l_onoff, lptr->l_linger);
    return(strres);
}

static char *
sock_str_timeval(union val *ptr, int len){
    struct timeval  *tvptr = &ptr->timeval_val;

    if (len != sizeof(struct timeval))
        snprintf(strres, sizeof(strres),
                 "size (%d) not sizeof(struct timeval)", len);
    else
        snprintf(strres, sizeof(strres), "%ld sec, %ld usec",
                 tvptr->tv_sec, tvptr->tv_usec);
    return(strres);
}
//(main code)-----------------------------------------------------------------------------------------------------------------
int
main(int argc, char **argv){

    int  listenfd, *iptr;
    pthread_t   tid;
    socklen_t   addrlen, len;
    struct sockaddr_in   servaddr;
    struct sockaddr *cliaddr;
    struct sock_opts    *ptr;

    void    sig_chld(int);

```

```
194     listenfd = Socket(AF_INET, SOCK_STREAM, 0);
195
196     /*for (ptr = sock_opts; ptr->opt_str != NULL; ptr++) {
197         printf("%s: ", ptr->opt_str);
198         if (ptr->opt_val_str == NULL)
199             printf("(undefined)\n");
200         else {
201             len = sizeof(val);
202             if (getsockopt(listenfd, ptr->opt_level, ptr->opt_name,
203                            &val, &len) == -1) {
204                 perror("getsockopt error");
205                 //err_ret("getsockopt error");
206             } else {
207                 printf("default = %s\n", (*ptr->opt_val_str)(&val, len));
208             }
209         }
210     }*/
211
212     bzero(&servaddr, sizeof(servaddr));
213     servaddr.sin_family      = AF_INET;
214     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
215     servaddr.sin_port        = htons(SERV_TCP_PORT);
216
217     Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
218
219     Listen(listenfd, LISTENQ);
220
221     //check argument count and assigned argument value to variable
222     if (argc == 3){
223         m  = atoi(argv[1]);
224         max_idle_time = atoi(argv[2]);
225     }else{
226         err_quit("usage: server_snc argument error\n");
227     }
228
229     //  maximum time the server waits for other clients. This time is started once any Server thread
    receives an Client connection; valid numbers: 1 to 300 seconds.
230     if (1>max_idle_time || max_idle_time>300){
231         printf("Maximum idle time server gives for other clients should be between 1 to 300 seconds \n");
232         exit(1);
233     }
234     //  maximum number of Clients at each session; valid numbers: 1 to 10.
235     if (1>m || m>10){
236         printf("Maximum number of Clients at each session of the game 1 to 10 \n");
237         exit(1);
238     }
239
240     printf("\nMaximum idle time, clients: %d \n" , max_idle_time);
241     printf("Maximum number of Clients : %d \n" , m);
242
243     printf(" ____________________________________________________________________________\n");
244     printf("| \n");
245     printf("|                                Welcome to SNC!\n");
246     printf("|                                    Server \n");
247     printf("|____________________________________________________________________________\n\n");
248
249     while(1) {
250
251         len = sizeof (cliaddr);
252         iptr = (int *) Malloc(sizeof(int));
253         iptr = Accept(listenfd, (SA *) &cliaddr, &len);
254
255         //The sending maximum clients reached warning to the clients
256         char maxCli_warn[] = "\nSorry! Maximum client count reached. Try next time\n";
257
```

```
258         //handling the number of clients
259         if((cli_count ) == m){
260             printf("Maximum clients connected. Client %d Rejected \n",cli_count+1);
261             send(iptr,&maxCli_warn,strlen(maxCli_warn),0);
262             close(iptr);
263         }
264 
265         Pthread_create(&tid, NULL, &doit, iptr);
266     }
267 }
268 //(doit function)----------------------------------------------------------------------
    ---
269 static void *
270 doit(void *arg){
271 
272     int connfd,connfdc;
273     ssize_t n;
274     connfd = (int *) arg;
275     char line[MAXLINE],buffer[256];
276     char arg1[20],arg2[20],arg3[20], conn[10];
277     bool flag = false;
278 
279     pthread_mutex_lock(&counter_mutex);
280 
281     cli_count++;//new client
282 
283     sprintf(conn, "%d", (int *)arg);
284     printf("Client : new connfd -");
285     strcpy(connfd_c[cli_count-1], conn);//add new client connfd to array
286     printf("%s\n",connfd_c[cli_count-1]);//print new client connfd
287 
288     pthread_mutex_unlock(&counter_mutex);
289 
290     //Pthread_detach(pthread_self());
291 
292     while(1) {
293 
294         time_t  ticks;
295         int yes = 1;
296 
297         if ( (n = Readline(connfd, line, MAXLINE)) == 0)
298             return;     /* connection closed by other end */
299 
300         pthread_mutex_lock(&counter_mutex);
301         sscanf(line,"%s %s %s",&arg1,&arg2,&arg3);//divide bufeer strig value from space
302         pthread_mutex_unlock(&counter_mutex);
303 
304         //join new client to server(working some functions)
305         if (strcasecmp(arg1, "JOIN") == 0 && flag == false){
306             pthread_mutex_lock(&counter_mutex);//mutex lock for data scrambaling
307 
308             //check simmilar realname
309             for(int z=0; z<m; z++){
310                 if(strcasecmp(arg3, realname[z]) == 0){
311                     bzero(buffer,sizeof(buffer));//empty buffer for new values
312                     sprintf(buffer, "Server : realname is already taken, try again!\n");
313                     Writen(connfd, buffer, sizeof(buffer));
314                     break;
315                 }
316             }
317 
318             strcpy(nickname[cli_count-1], arg2);//add nickname to array
319             strcpy(realname[cli_count-1], arg3);//add realname to array
320             flag = true;//stop join create multiple nicknames in same client
321 
```

```
            //welcome message
            bzero(buffer,sizeof(buffer));
            sprintf(buffer, "Server : %s, welcome to SNC!\n", nickname[cli_count-1]);//welcome message to client
            write(connfd, buffer, sizeof(buffer));

            //send wilcome message to other clients
            //send_message_all(nickname[cli_count-1], cli_count, conn);

            pthread_mutex_unlock(&counter_mutex);
        }
        //check clent real name using nickname(working)
        else if (strncasecmp(arg1, "WHOIS",5) == 0){
            pthread_mutex_lock(&counter_mutex);

            int z=0;
            while(1){
                if(z < m){
                    //check nick name is in chatroom
                    if(strcasecmp(arg2, nickname[z]) != 0){
                        bzero(buffer,sizeof(buffer));//empty buffer for new values
                        sprintf(buffer, "Server : nickname is not in Simple Network Chat (SNC)!\n");
                        Writen(connfd, buffer, sizeof(buffer));
                        break;
                    }else if(strcasecmp(nickname[z], arg2) == 0){
                        bzero(buffer,sizeof(buffer));
                        sprintf(buffer, "Server : %s %s\n", nickname[z],realname[z]);
                        write(connfd, buffer, sizeof(buffer));
                        break;
                    }
                }
                z++;
            }

            pthread_mutex_unlock(&counter_mutex);
        }
        //check time and date(working)
        else if (strncasecmp(arg1, "TIME", 4) == 0){
            pthread_mutex_lock(&counter_mutex);

            time(&ticks);
            bzero(buffer,sizeof(buffer));
            snprintf(buffer, sizeof(buffer), "Server : %.24s\r\n", ctime(&ticks));
            write(connfd, buffer, sizeof(buffer));

            pthread_mutex_unlock(&counter_mutex);

        }
        //quit from client(woking)
        else if (strncasecmp(arg1, "QUIT",4) == 0){
            printf("test quit 1");
            pthread_mutex_lock(&counter_mutex);
            printf("test quit");

            bzero(buffer,sizeof(buffer));
            printf("Client is quiting ... \n");
            sprintf(buffer, "Server : Bye %s\n", nickname[cli_count-1]);
            write(connfd, buffer, sizeof(buffer));

            pthread_mutex_unlock(&counter_mutex);
        }
```

```
382         //quit from server and client(working)
383         else if (strncasecmp(arg1, "QUITSER",7) == 0){
384             pthread_mutex_lock(&counter_mutex);
385 
386             bzero(buffer,sizeof(buffer));
387             printf("server and client are quiting ... \n");
388             sprintf(buffer, "Server : Server is terminated\n");
389             write(connfd, buffer, sizeof(buffer));
390             sleep(1);
391             close(connfd);
392             exit(1);
393             pthread_mutex_unlock(&counter_mutex);
394 
395         }
396         //reset idle time value in client
397         else if(strncasecmp(arg1, "ALIVE",5) == 0){
398             pthread_mutex_lock(&counter_mutex);
399             pthread_mutex_unlock(&counter_mutex);
400         }
401         //send message to another user using nickname
402         else if(strncasecmp(arg1, "MSG",3) == 0){
403             pthread_mutex_lock(&counter_mutex);
404 
405             int w=0;
406             while(1){
407                 if(w < m){
408                     //check nick name is in chatroom
409                     if(strcasecmp(arg2, nickname[w]) != 0){
410                         bzero(buffer,sizeof(buffer));//empty buffer for new values
411                         sprintf(buffer, "Server : nickname is not in Simple Network Chat (SNC)!\n");
412                         Writen(connfd, buffer, sizeof(buffer));
413                         break;
414                     }else if(strcasecmp(nickname[w], arg2) == 0){
415                         send_message_client(nickname[w],cli_count,arg3);//call function to send message
416                         //bzero(buffer,sizeof(buffer));
417                         //sprintf(buffer, "%s : %s\n", nickname[w], arg3);//output nickname : message
418                         break;
419                     }
420                 }
421                 w++;
422             }
423 
424             pthread_mutex_unlock(&counter_mutex);
425         }
426         //SNC command error
427         else{
428             if(flag == true){
429                 bzero(buffer,sizeof(buffer));
430                 sprintf(buffer, "Server : cannot create multiple nickname in one client program!\n");
431                 write(connfd, buffer, sizeof(buffer));
432             }else{
433                 bzero(buffer,sizeof(buffer));
434                 sprintf(buffer, "Server : SNC command is wrong, try again!\n");
435                 write(connfd, buffer, sizeof(buffer));
436             }
437         }
438 
439         Writen(connfd, line, n);
440 
441         /* Reduce CPU usage */
442         sleep(1);
443     }
444 
445     Close(connfd);          /* we are done with connected socket */
446     return(NULL);
447 }
```

“Server_snc.c” is the main file that acts as a router for another client to communicate between them. A few global variables will store the maximum allowed client, nickname, realname, etc, for data saving.

- Line (10): Initialize pthread mutext for lock and unlock codes that multiple clients use simultaneously.
- Line (13 – 14): Include a structure that will be used to calculate timeout.
- Line (21 - 24): Static variables that will use in timeout.
- Line (26 - 31): Structure for saving socket data.
- Line (33 - 64): include socket options and can print in the server console.
- Line (72 - 84):” queue_remove()” function will remove the client that quit the chatroom from the structure(use array instead of structure for getting output).
- Line (87 - 102): “send_message_all()” will send a message to all clients in the chatroom except for the sender.
- Line (108 - 127): “send_message_client()” will send a message to the desired client(client will find using the nickname entered by the user).
- Line (170 -180): “sock_str_timeval” use in idle time calculation.

The main function first runs before those above functions are called.

- Line (196 -210): set socket and get socket number as “listenfd”.
- Line (212 - 215): save socket data in “servaddr” structure.
- Line (217): bind “servaddr” for socket.
- Line (219): listen for SYN to start a three-way handshake.
- Line (222 - 227): check the argument that the user typed and check for validity and assign those arguments for “m” and “max_idle_time”.
- Line (230 - 233): check max client within the range of 1 to 10
- Line (235 - 238): check “max_idle_time” within range 1 to 300
- Line (240 - 247): print “m”, “max_idle_time”, welcome message.

Then we have an infinite while loop to check client count within maximum value and then create a thread for the Server and call doit within this thread.

“doit()” function will have most of the code that checks client conditions.

- Line (272 - 277) : in function variable that use for “doit()” function.
- Line (281): client count will increase by 1.
- Line (283 - 286): client connfd will print in server console.

Then infinite while loop has if-else conditions to check client SYN commands.

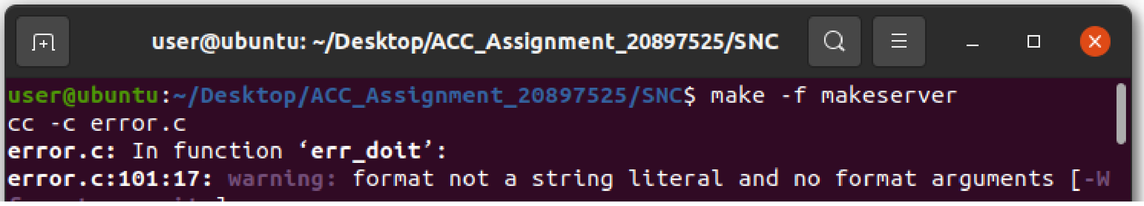
- Line (297): read connfd and save a string to line char.
- Line (301): line sting value will divide into three parts for value comparison.
- Line (305 - 331): check client typed “JOIN” in the client console and review for a similar nickname in the chat room. If there is no nickname and real name will be added to the 2D array for later use. and the Server sends the client a welcome message (use write() function) and runs “send_message_all()” to say a new user has entered the chat room.
- Line (333 - 356): check the client type “WHOIS” and print the nickname and real name after reviewing the given nickname in the array, or else the Server will send a message to the client for the wrong nickname.
- Line (357 – 368): check client type “TIME” and print server data and time for a requested client using write() function.
- Line (369 - 381): check client type “QUIT”, and the client will close the client socket, and the server side will send to the client saying “Server : buy <nickname>” (message sending was stopped working after adding new code parts and only client stop without a message).

- Line (383 - 395): check client type “QUITSERVER”, and the Server will close along with all other clients. (stop working after adding/modifying new code).
- Line (396 - 400): check client type “ALIVE” and reset the client given time to the initial value. (Not coded)
- Line (401 - 425): check the client type “MSG” and find a nickname in the array or send the requested client to retry using another nickname that is in the chat room (working). And if a nickname is in the array, it will call “send_message_client()” to send a specified client with a message.
- Line (426 - 437): else condition has an if the condition that will check flag variable (Boolean) is “true” client cannot use “JOIN” command again to create a new nickname and real name, or else the client will get an output that SNC command is wrong to try again.

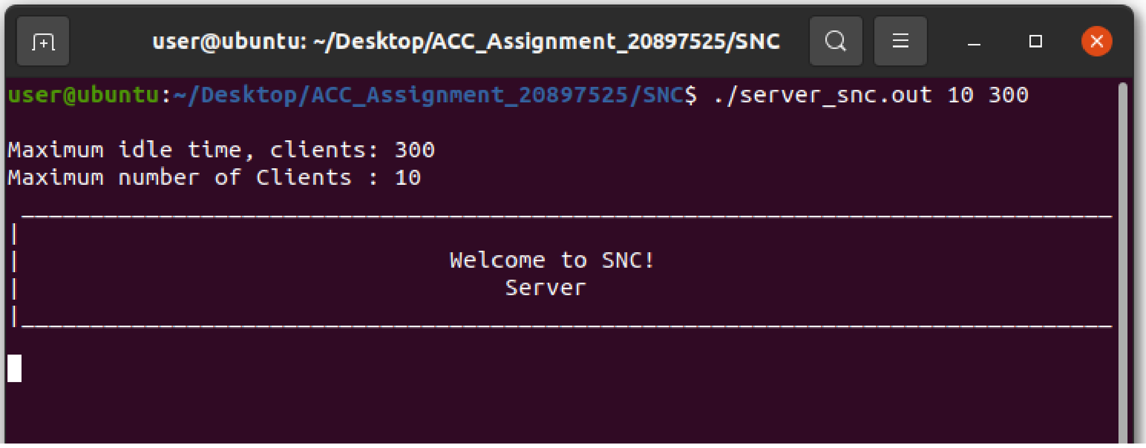
❖ Other than the above files, some other files from the lab worksheets have been used.

How to compile your program and how to run the program

```
+-----------------------------------------------------------------------------------+
|          Simple Network Chat (SNC) Instructions to run server and client programs
+-----------------------------------------------------------------------------------+
|
+---> How to run server-side
|          |
|          +--> Create the executable using the make functions
|          |          |
|          |          +--> command: make -f makeserver
|          |
```

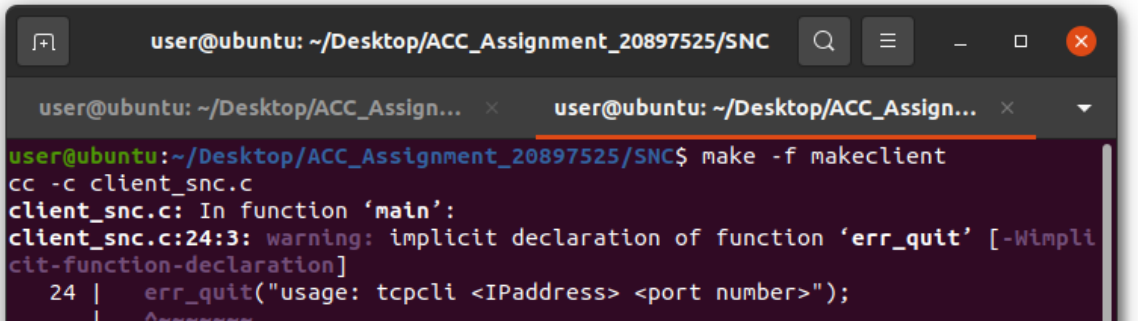


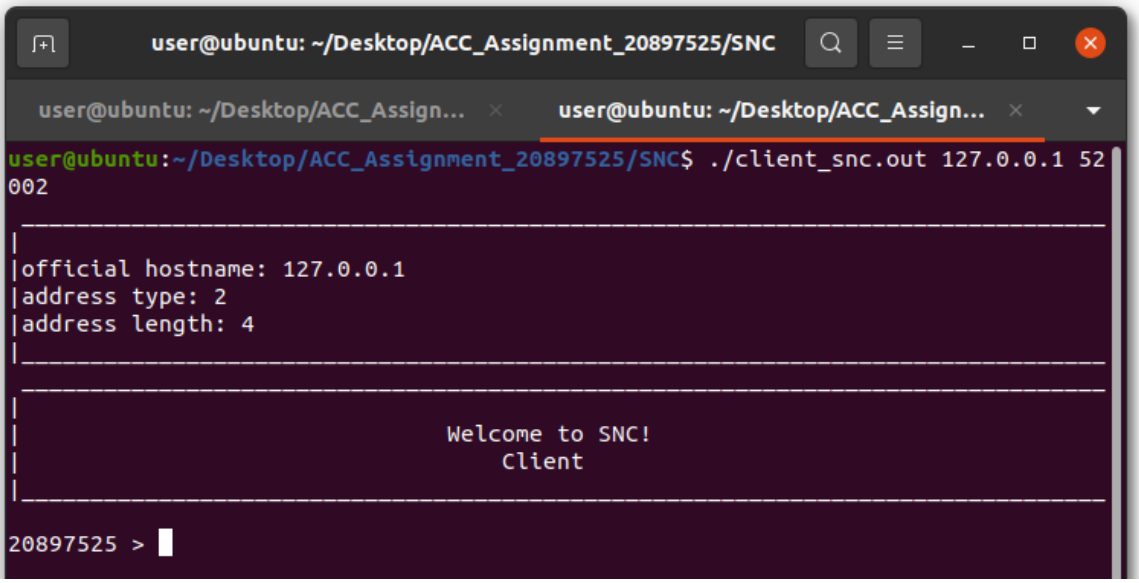

```
|          |
|          Run the executable file
|                     |
|                     +--> command: ./server_snc.out [m] [max_wait_game]
|                                |
|                                +--> [max_wait_game]   - Maximum time the server waits
|                                +--> [m]- Maximum number of MClients
|
```



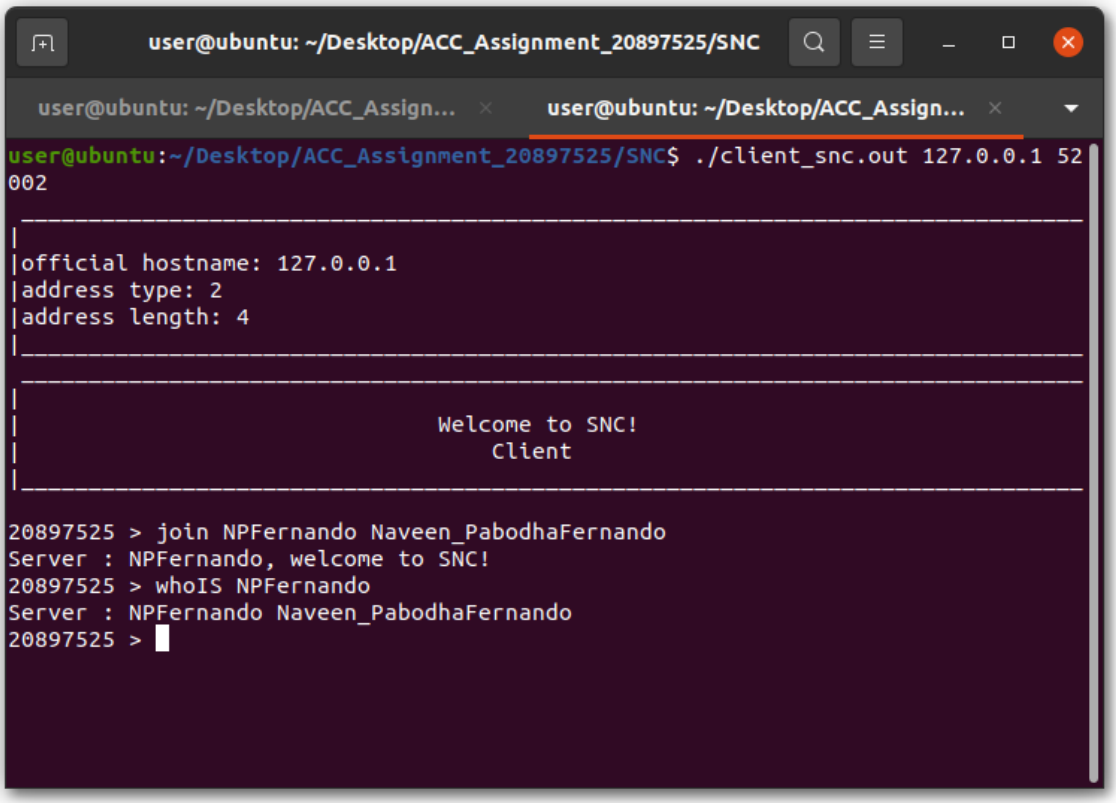

```
|
|
|
```

|
+---> How to run the client side
|
+--> Create the executable using the make functions
| |
| +--> command: make -f makeclient
|
|
|
|
|
|
|
|
|
|
|
|
+--> Run the executable file
|
+-->command: ./client_snc.out 127.0.0.1 52001
|
+--> 127.0.0.1 - IP address of server
+--> 52002 - port number

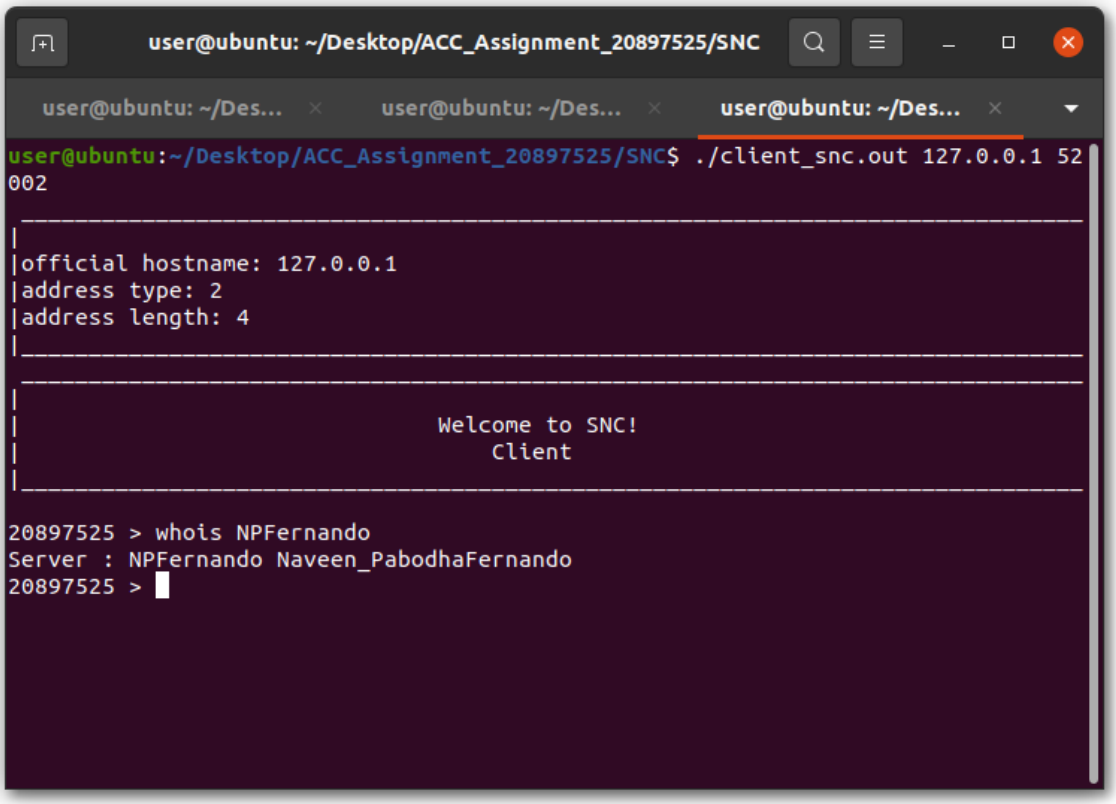




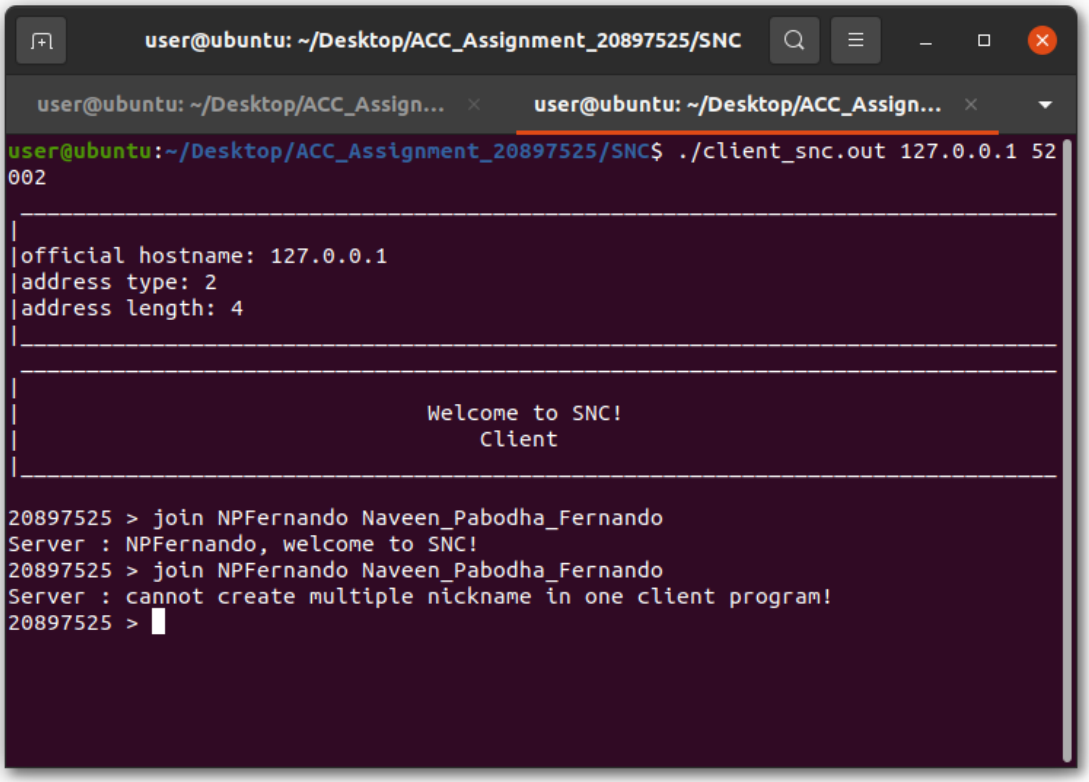
Sample inputs and outputs from running your programs



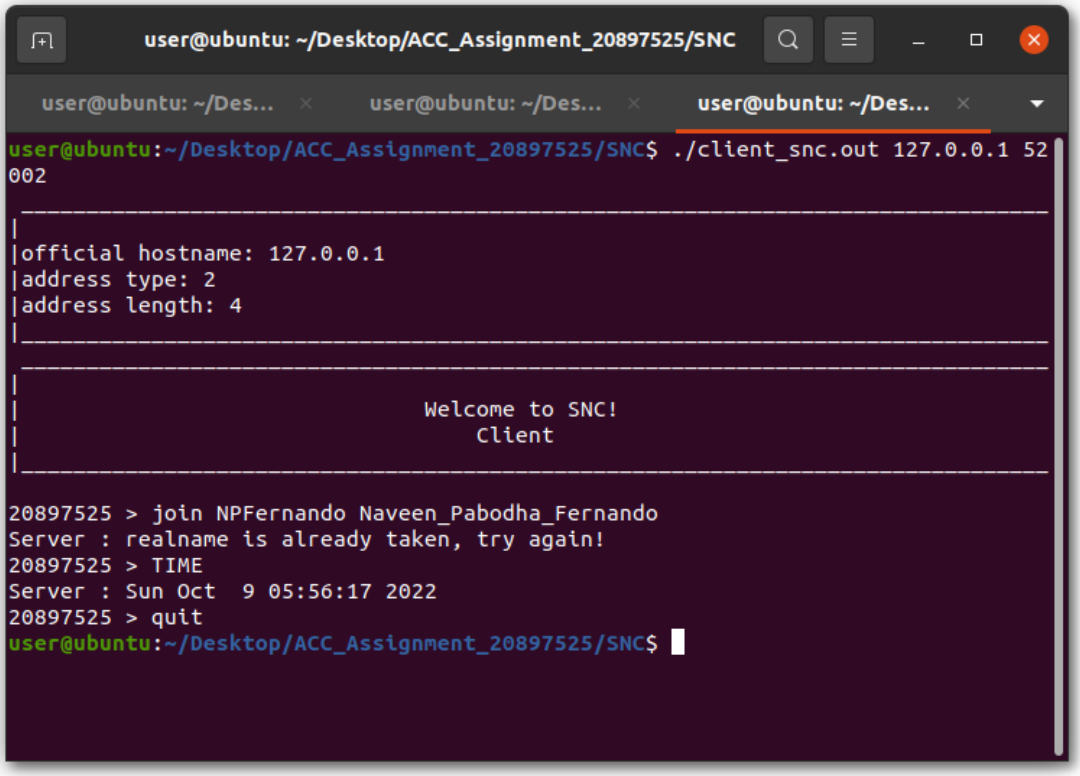
"JOIN" will add a new user to the Server, and "WHOIS" can get the user real name using nick name.



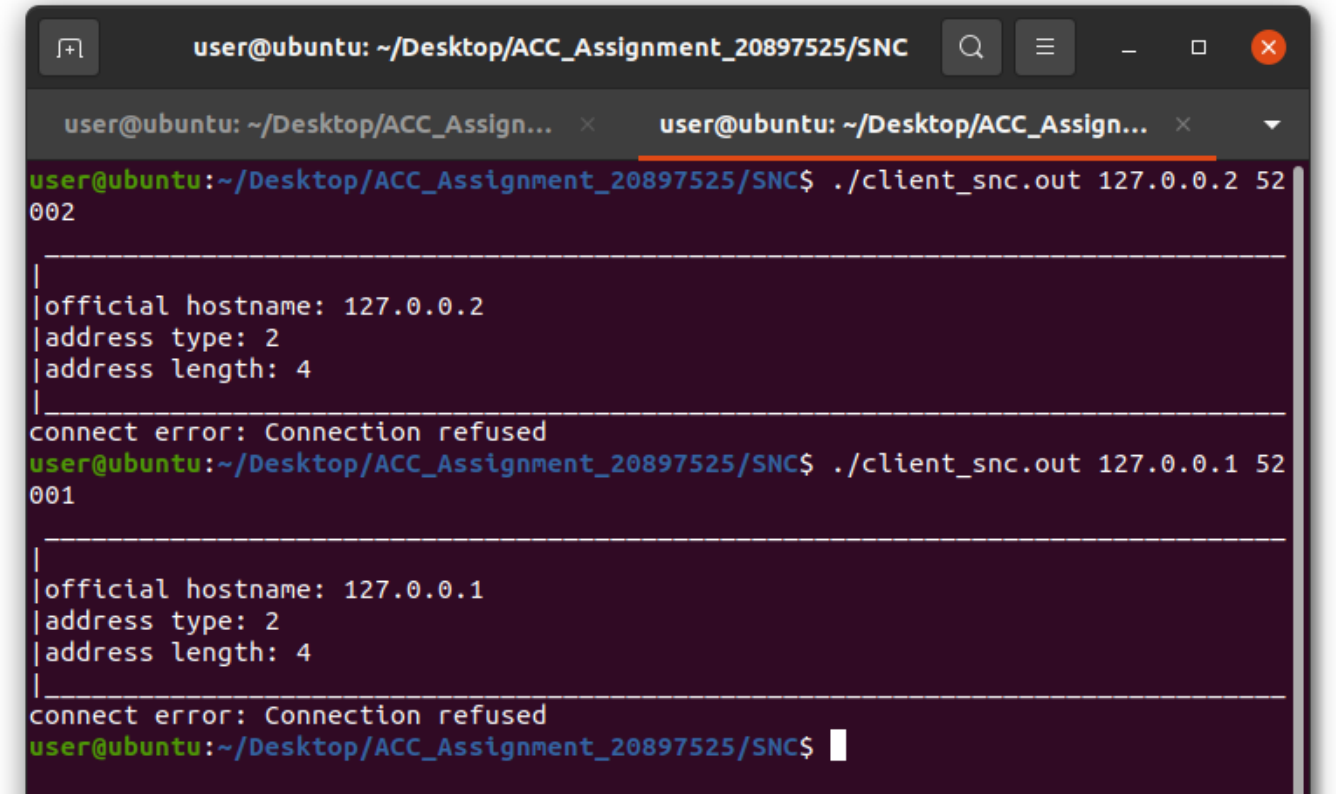
When the user types "WHOIS" in another client console using a nickname or other client request, the client can get the real name from the Server.

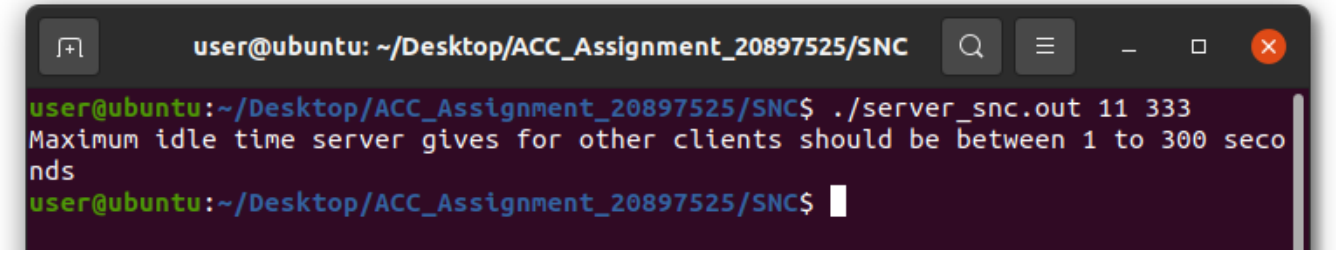


In one client, “JOIN” can only be used once or get an error message.



The client can check the time using the “TIME” command to request time from the server side, and the Server will send the date and time for the requested client.





When arguments are not within limit (“m” > 10, 1 > “max_idle_time” > 300) or port number or IP address is wrong connection will terminate.

Cases for which your program is not working correctly

```
85
86 //Send message to all clients except sender(code or structure error)
87 void send_message_all(char *nickname, int cli_count, char conn){
88     pthread_mutex_lock(&counter_mutex);
89
90     for(int i=0; i<m; ++i){
91         if(strcasecmp(nickname[i], nickname[cli_count-1])){
92             if(strcmp(connfd_c[i],conn) != 0){
93                 bzero(buffer,sizeof(buffer));
94                 sprintf(buffer, "Server : hi 3!\n");
95                 write(conn, buffer, sizeof(buffer));
96                 if(write(connfd_c[i], nickname, strlen(nickname)) < 0){
97                     perror("ERROR: write to descriptor failed");
98                     break;
99                 }
100            }
101        }
102    }
103
104    pthread_mutex_unlock(&counter_mutex);
105 }
106
107 //Send message to client using nickname(code or structure error)
108 void send_message_client(char *nickname, int cli_count, char *msg){
109     pthread_mutex_lock(&counter_mutex);
110
111     int conn;
112     for(int i=0; i<m; ++i){
113         //cheak nickname
114         if(strcasecmp(nickname[i], nickname)){
115             //conn need to replce to nickname client connfd
116             if(strcmp(connfd_c[i],conn) == 0){
117                 bzero(buffer,sizeof(buffer));
118                 sprintf(buffer, "%s : %s\n",nickname, msg);//message save in buffer
119                 write(conn, buffer, sizeof(buffer));//write buffer to client
120                 if(write(connfd_c[i], nickname, strlen(nickname)) < 0)
121                 {
122                     perror("ERROR: write to descriptor failed");
123                     break;
124                 }
125             }
126         }
127     }
128
129     pthread_mutex_unlock(&counter_mutex);
130 }
```

- "send_message_all()" and "send_message_client()" functions are not properly working due to structure issue. But code is coded for work with arrays.

```
369         //quit from client(woking)
370         else if (strncasecmp(arg1, "QUIT",4) == 0){
371             printf("test quit 1");
372             pthread_mutex_lock(&counter_mutex);
373             printf("test quit");
374
375             bzero(buffer,sizeof(buffer));
376             printf("Client is quiting ... \n");
377             sprintf(buffer, "Server : Bye %s\n", nickname[cli_count-1]);
378             write(connfd, buffer, sizeof(buffer));
379
380             pthread_mutex_unlock(&counter_mutex);
381         }
```

- "QUIT" was adequately working and returned Server: Buy <nickname> message to the client, but after adding later code to Server, it stopped printing buffer data to the client, but the client stopped after "QUIT" without the message. Could not find the problem.

```
401         //send message to another user using nickname
402         else if(strncasecmp(arg1, "MSG",3) == 0){
403             pthread_mutex_lock(&counter_mutex);
404 
405             int w=0;
406             while(1){
407                 if(w < m){
408                     //check nick name is in chatroom
409                     if(strcasecmp(arg2, nickname[w]) != 0){
410                         bzero(buffer,sizeof(buffer));//empty buffer for new values
411                         sprintf(buffer, "Server : nickname is not in Simple Network Chat (SNC)!\n");
412                         Writen(connfd, buffer, sizeof(buffer));
413                         break;
414                     }else if(strcasecmp(nickname[w], arg2) == 0){
415                         send_message_client(nickname[w],cli_count,arg3);//call function to send message
416                         //bzero(buffer,sizeof(buffer));
417                         //sprintf(buffer, "%s : %s\n", nickname[w], arg3);//output nickname : message
418                         break;
419                     }
420                 }
421                 w++;
422             }
423 
424             pthread_mutex_unlock(&counter_mutex);
425         }
```

- “MSG” does not work with 2D arrays, but it can check the nickname in the array. If previous structure code was worked before using arrays code could have get MSG function working with other clients for send message to specific client.

References

1. Worksheets 1, 4, 5(modified using Worksheets 5), and six under the ACC module.