CS425: Computer Networks
IIT Kanpur
Project 3: Designing a STCP Transport Layer
Date: Wed, Sep 30

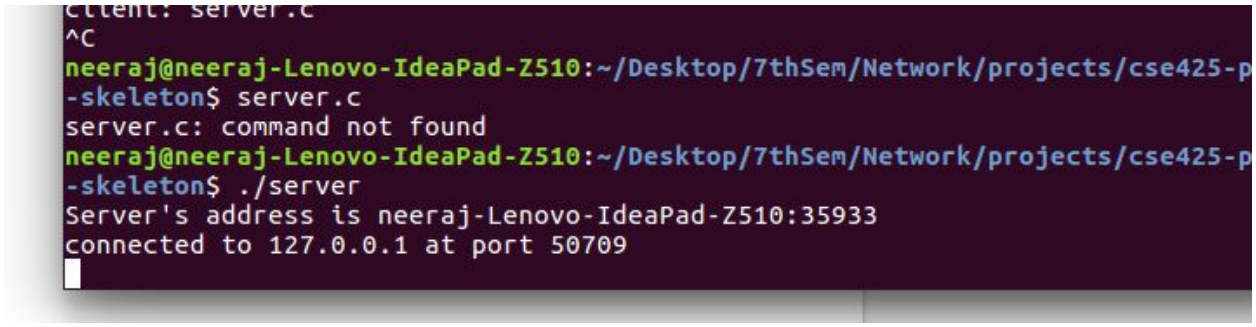
Name: Neeraj Kumar
Roll: 13427
Email: neerajkr@iitk.ac.in
Dept: EE

Design Choices:

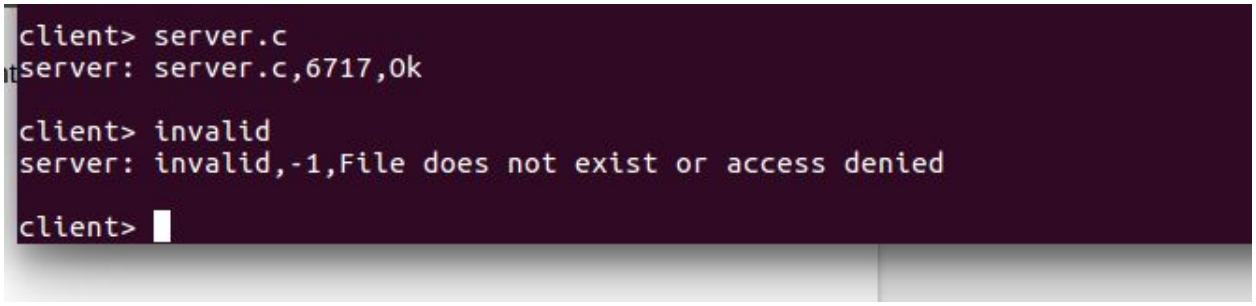
- Firstly, i implemented 3 way handshake for establishing connection between server and client side.
- After, establishing the connection i checked the event for “APP_DATA”, “NETWORK_DATA” and “APP_CLOSE_REQUESTED” in else if loops inside function “control_loop()”.
- If there is an event from application layer then, i received the data and attached a STCP header to it and sent it to network layer, taking care of window size ,sequence number and acknowledgement number.
- In the second else if for “NETWORK_DATA”, i received the data from application layer and then extracted header and data(if present) from it. I checked the flags present in header and made cases for different kind of flags and handled all the cases separately.
- In the third and last else if for “APP_CLOSE_REQUESTED”, i sent the FIN flag to application layer because this request will be generated when either server or client want to close the connection.
    - Since, for closing the connection, FIN flag is sent as well as received before closing the connection, so i have made two other connection state apart from “CSTATE_ESTABLISHED”- FIN_WAIT_1 & FIN_WAIT_2
- I have also taken care correct endianess with *htonl/ntohl* or *htons/ntohs* as mentioned.

Testing procedures and results:

- I compiled the make file and then typed "./server" on terminal. Then , it gives the ouptut- "Server's address is neeraj-Lenovo-IdeaPad-Z510:35933".
- I opened another terminal in same directory and typed "./client neeraj-Lenovo-IdeaPad-Z510:35933".
- Then A message on server side terminal comes, written- "connected to 127.0.0.1 at port 50709", indicating connection is establised. The screenshot is attached below for server and client side respectively.



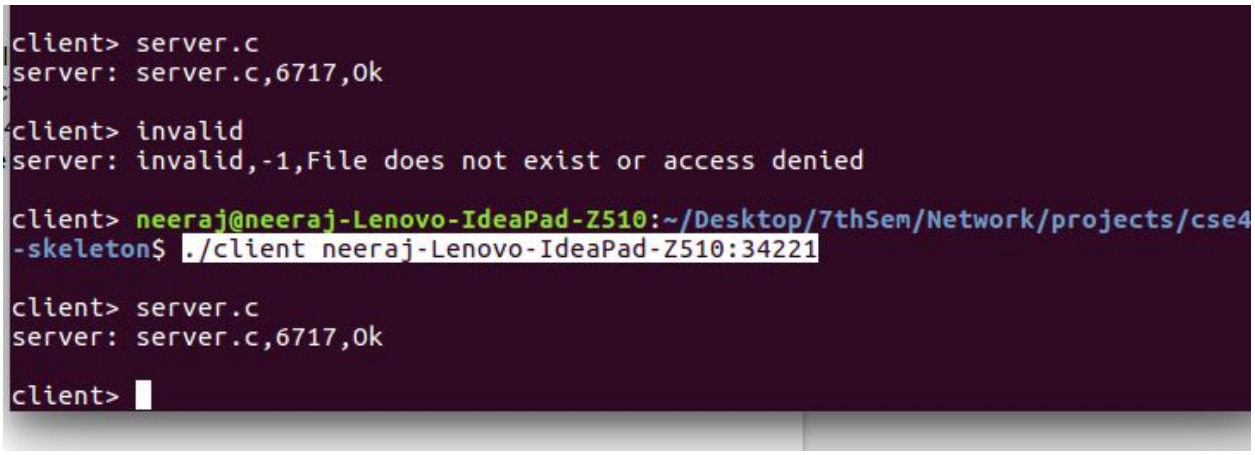
- Then , i typed filename on client terminal- say "server.c". Then , a status prints showing correct transfer of file and also on giving an incorrect file, it gives an error message. Screenshot is attached below for both the cases.

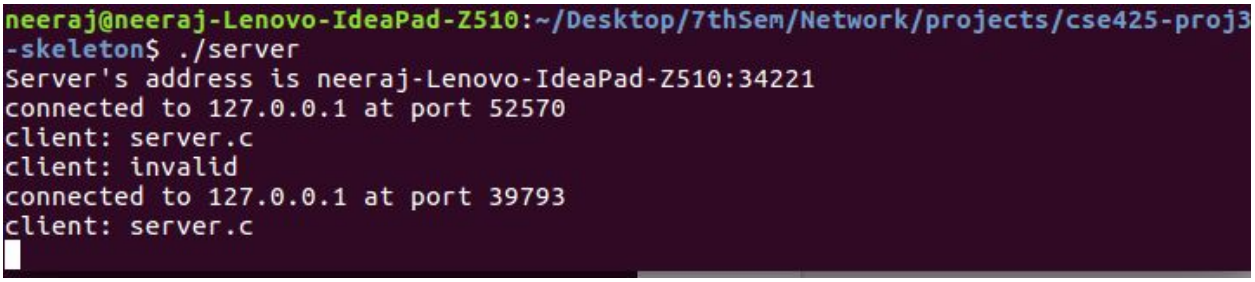


- I also checked the file size of server.c in the directory and it's length is 6717, indicating correct transfer of file.
- I also printed the content of file on terminal and it was same as original file content.

Closing the client:

- For closing the connection on client side, i clicked “ctrl+D” and then client connection closes. After closing the connection , again typed “./client neeraj-Lenovo-IdeaPad-Z510:34221” and connection established again. After connecting, second time, i gave the filename and the result was same as earlier. Below is the screenshot for this case on client side and server side respectively.





Summary Of test results:

- I have implemented all the metioned design requirements and they are running correctly.

Appendix:

Sourcefile Code:

Transport.c

```
  /*
 * transport.c
 *
 *  Project 3
 *
 * This file implements the STCP layer that sits between the
 * mysocket and network layers. You are required to fill in the STCP
 * functionality in this file.
 *
 */


#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <arpa/inet.h>
#include "mysock.h"
#include "stcp_api.h"
#include "transport.h"

 #define MAX_WIN_SIZE 3072
```

```
enum { CSTATE_ESTABLISHED ,FIN_WAIT_1,FIN_WAIT_2};    /* you should have more
states */

STCPHeader A,B,C,D,E,F;       //These STCP headers are for handshaking between client and
server
/* this structure is global to a mysocket descriptor */

int AcknowledgedByte,NextExpectedByte,SequenceNumber;       //Global variables to handle
window size, sequnce number and acknowledgemnt number

typedef struct
{
    bool_t done;    /* TRUE once connection is closed */

    int connection_state;   /* state of the connection (established, etc.) */
    tcp_seq initial_sequence_num;

    /* any other connection-wide global variables go here */
} context_t;


static void generate_initial_seq_num(context_t *ctx);
static void control_loop(mysocket_t sd, context_t *ctx);


/* initialise the transport layer, and start the main loop, handling
 * any data from the peer or the application.  this function should not
 * return until the connection is closed.
 */
void transport_init(mysocket_t sd, bool_t is_active)
{
    context_t *ctx;

    ctx = (context_t *) calloc(1, sizeof(context_t));
    assert(ctx);

    generate_initial_seq_num(ctx);

    if(is_active) {                  //client side during handshake
        A.th_flags=TH_SYN;               //sending SYN flag from client
        A.th_seq=htonl(rand()%256);              //Initial sequence Number from client side
        stcp_network_send(sd, &A, sizeof(A),NULL);          //SYN flag is sent to network layer to
trasmit finally to server
```

```
        stcp_network_recv(sd, &B, sizeof(B));            //recieved ACK & SYN flag from server
        E.th_flags=TH_ACK;                               //ACK flag to acknowledge the received SYN
from server
        E.th_seq=B.th_ack;
        E.th_ack=B.th_seq+1;                             //acknowledgement and sequence number is
being transferred to server
        stcp_network_send(sd, &E, sizeof(E),NULL);       //STCP header containg ACK is sent
to network layer
        AcknowledgedByte=B.th_ack-1;                     //Initialisng global varibled used during
data transmission
        SequenceNumber=B.th_ack;
        NextExpectedByte=E.th_ack;
    }


    else{          //server side

        stcp_network_recv(sd, &C, sizeof(C));            //SYN flag recieved from client
        D.th_seq=htonl((rand()+20)%256);                 //Initial sequence Number from server
side
        D.th_flags=TH_ACK|TH_SYN;                        //ACK and SYN flags to acknowledge the
received SYN from client and sent another SYN
        D.th_ack=C.th_seq+1;
        stcp_network_send(sd, &D, sizeof(D),NULL);       //STCP header sent to network layer
for transimission to client side
        stcp_network_recv(sd,&F,sizeof(F));
        AcknowledgedByte=D.th_seq;                       //Initialising the gloabl varibales on server
side
        SequenceNumber=E.th_ack;
        NextExpectedByte=D.th_ack;

    }

    /* XXX: you should send a SYN packet here if is_active, or wait for one
     * to arrive if !is_active.  after the handshake completes, unblock the
     * application with stcp_unblock_application(sd).  you may also use
     * this to communicate an error condition back to the application, e.g.
     * if connection fails; to do so, just set errno appropriately (e.g. to
     * ECONNREFUSED, etc.) before calling the function.
     */
    ctx->connection_state = CSTATE_ESTABLISHED;
    stcp_unblock_application(sd);
```

```
    control_loop(sd, ctx);

    /* do any cleanup here */


    free(ctx);
}


/* generate random initial sequence number for an STCP connection */
static void generate_initial_seq_num(context_t *ctx)
{
    assert(ctx);

#ifdef FIXED_INITNUM
    /* please don't change this! */
    ctx->initial_sequence_num = 1;
#else
    /* you have to fill this up */
    ctx->initial_sequence_num =rand()%256;
#endif
}


/* control_loop() is the main STCP loop; it repeatedly waits for one of the
 * following to happen:
 *   - incoming data from the peer
 *   - new data from the application (via mywrite())
 *   - the socket to be closed (via myclose())
 *   - a timeout
 */

static void control_loop(mysocket_t sd, context_t *ctx)
{
    assert(ctx);
    assert(!ctx->done);

    char AppBuffer[STCP_MSS],NetworkBuffer[556];          //DATA buffers for storing during
transmission
    STCPHeader Header,FIN_Header,Network_FIN;             //STCP headers used during
different events described below

    while (!ctx->done)
```

```
  {
     Header.th_win=MAX_WIN_SIZE;                //MAX_WIN_SIZE is 3072 bytes for flow
control
     unsigned int event;
     event = stcp_wait_for_event(sd, ANY_EVENT, NULL);       //ANY_EVENT to handle events
from app and network layer

     /* check whether it was the network, app, or a close request */
     if (event & APP_DATA)       //If there is a request from app layer then add header to it and
transfer to network layer
     {

        int BytesReceivedFromAppLayer=stcp_app_recv(sd, AppBuffer,
MIN(Header.th_win,STCP_MSS));    //receiving data from APP layer
        Header.th_seq=htonl(SequenceNumber);
        Header.th_ack=htonl(NextExpectedByte);
        Header.th_win=htons(Header.th_win+BytesReceivedFromAppLayer);

        SequenceNumber+=BytesReceivedFromAppLayer;
        int
StatusSentToNetworkLayer=stcp_network_send(sd,&Header,sizeof(Header),&AppBuffer,
BytesReceivedFromAppLayer,NULL);
           //sending data to network layer after attaching headers to it of proper flags and
ack,seq,win numbers
     }

     else if(event & NETWORK_DATA ){     //handling request from networl layer

        int BytesReceivedFromNetworkLayer=stcp_network_recv(sd,NetworkBuffer,556);  //recv
from network layer
        char DataExcludingHeader[BytesReceivedFromNetworkLayer-20];
        strncpy ( DataExcludingHeader, NetworkBuffer+20,
BytesReceivedFromNetworkLayer-20);   //extract data descarding header
        memcpy(&Header, NetworkBuffer, 20); //extract header

        if (Header.th_flags==TH_ACK){       //if ACK flag is present then, increase window size
and modify Ackneldged Byte
           Header.th_win=ntohl(Header.th_ack)-AcknowledgedByte+ntohs(Header.th_win);
           AcknowledgedByte=ntohl(Header.th_ack);
        }

        STCPHeader TempHeader;
```

```
        if (BytesReceivedFromNetworkLayer>20){      //if msg from network layer contains data
as well then trasfer data to app layer
            printf("%s", DataExcludingHeader);
            stcp_app_send(sd, DataExcludingHeader, BytesReceivedFromNetworkLayer-20);
            NextExpectedByte+=BytesReceivedFromNetworkLayer-20;
            TempHeader.th_flags=TH_ACK;

            TempHeader.th_ack=htonl(NextExpectedByte);
            TempHeader.th_seq=htonl(SequenceNumber);

            stcp_network_send(sd, &TempHeader,20, NULL);    //send the ACK to network layer
for received data
        }

        if ((Header.th_flags == TH_FIN) || (Header.th_flags == TH_FIN+TH_ACK))  { //IF FIN is
present alone or with ACK flags
            Network_FIN.th_seq = htonl(SequenceNumber);
            Network_FIN.th_ack = htonl(NextExpectedByte);
            Network_FIN.th_flags = TH_ACK;
            stcp_network_send(sd, &Network_FIN, 20, NULL);      //send ACK of FIN to network
layer

            if(ctx->connection_state == FIN_WAIT_1)     //Check two other states whether FIN
has been sent as well as receivd or not
                ctx->done = true;
            else
                ctx->connection_state = FIN_WAIT_2;
            stcp_fin_received(sd);
        }

    }

    else if (event & APP_CLOSE_REQUESTED){          //If there is a request for closing the
connection from app layer
        FIN_Header.th_flags=TH_FIN;         //send the FIN flag to network layer
        FIN_Header.th_seq=htonl(SequenceNumber);
        FIN_Header.th_ack=htonl(NextExpectedByte);
        FIN_Header.th_win=htons(Header.th_win);
        stcp_network_send(sd, &FIN_Header, 20, NULL);
        stcp_network_recv(sd, &FIN_Header, 20);     //receive ACK of sent FIN flag from
network layer
        if(ctx->connection_state == FIN_WAIT_2)     //check for both the states, whether to close
or not
```

```
            ctx->done = true;
        else
            ctx->connection_state = FIN_WAIT_1;
    }

  }
}


/*********************************************************************/
/* our_dprintf
 *
 * Send a formatted message to stdout.
 *
 * format               A printf-style format string.
 *
 * This function is equivalent to a printf, but may be
 * changed to log errors to a file if desired.
 *
 * Calls to this function are generated by the dprintf amd
 * dperror macros in transport.h
 */

void our_dprintf(const char *format,...)
{
   va_list argptr;
   char buffer[1024];
   assert(format);
   va_start(argptr, format);
   vsnprintf(buffer, sizeof(buffer), format, argptr);
   va_end(argptr);
   fputs(buffer, stdout);
   fflush(stdout);
}
```