COMPUTER NETWORKS(CS425A)

PROJECT 3:

30th September, 2016

KRITI JOSHI - 13358

kritij@iitk.ac.in

IMPLEMENTED OPTIONS AND DESIGN CHOICES

1) Implemented STCP protocol.
2) 3-way handshake for initiation
3) Handled Endianness(ntohl,htonl)
4) Wrap data from app in proper header and send to network layer
5) Obtain data from network, remove header and send to app
6) Proper offset to read and write data
7) Implemented proper 4 step termination of connection
8) Handled Ctrl-C in client
9) Handled packets which didn't fit in window (Other type of responses i.e. anything accept APP_DATA flag was processed till an acknowledgment request was seen)
10) Handled FIN with and without data

TESTING METHODS:

1) Code was properly tested on the given server and client files (with -f and without -f option)
2) Other things like Ctrl-C were also tested

RESULTS:

Code worked properly for the above methods. “Rcvd” file similar to the input file could be obtained.

APPENDIX:

```
/*
 * 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"
#include <time.h>
#include <stdlib.h>
#include <math.h>

// #define min(a,b) (a<b?a:b)
// #define OFF (uint8_t)140
enum { CSTATE_ESTABLISHED };    /* you should have more states */


/* this structure is global to a mysocket descriptor */
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;
    srand(time(NULL));
    ctx = (context_t *) calloc(1, sizeof(context_t));
    assert(ctx);
    uint8_t OFF = 5;
    generate_initial_seq_num(ctx);

    /* 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.
     */
     // client
     if(is_active){
        // printf("Active\n");
        int flag=0;
        tcphdr stcpHeader;
        tcphdr* synAckHeader;
        // set seqx and SYN flag
        stcpHeader.th_seq = htonl(ctx->initial_sequence_num);
        stcpHeader.th_off = OFF;
        stcpHeader.th_flags = TH_SYN;

        // send header to server
        int sentData = stcp_network_send(sd, &stcpHeader, sizeof(stcpHeader),NULL); //
myaccept ran
        if(sentData<=0){
            printf("Send failed");
            flag=1;
            goto label;
        }
        // wait for SYN_ACK packet
        synAckHeader=(tcphdr*)malloc(STCP_MSS);
        stcp_wait_for_event(sd, NETWORK_DATA, NULL);
        stcp_network_recv(sd,synAckHeader,STCP_MSS);
        if(synAckHeader->th_flags==(TH_SYN|TH_ACK)){
```

```
            tcphdr ackHeader;
            // for pure ack packet, ack num = next unsent seq numberss
            ackHeader.th_seq=htonl(ctx->initial_sequence_num+1);
            ackHeader.th_ack = htonl(ntohl(synAckHeader->th_seq)+1);
            ackHeader.th_off=OFF;
            ackHeader.th_flags=TH_ACK;
            sentData = stcp_network_send(sd,&ackHeader,sizeof(ackHeader));
            if(sentData<=0){
                printf("Send failed");
                flag=1;
            }
        }else{
            flag=1;
        }
        label:

        // set errno
        if(flag){
            errno=ECONNREFUSED;
        }
        free(synAckHeader);
    }else{ // server
        // printf("Passive\n");
        tcphdr* ackHeader;
        int flag=0;
        //get SYN header
        tcphdr* stcpHeader=(tcphdr*) malloc(STCP_MSS);
        stcp_network_recv(sd,stcpHeader,STCP_MSS);

        // check SYN flag
        if(stcpHeader->th_flags==TH_SYN){
            // create syn-ack packet
            tcphdr synAckHeader;
            synAckHeader.th_seq=htonl(ctx->initial_sequence_num);
            synAckHeader.th_ack=htonl(ntohl(stcpHeader->th_seq)+1);
            synAckHeader.th_flags=TH_ACK|TH_SYN;
            synAckHeader.th_off=OFF;
            int sentData = stcp_network_send(sd, &synAckHeader,
sizeof(synAckHeader),NULL);
            if(sentData<=0){
                printf("Send failed");
                flag=1;
                goto label2;
            }
            // waits for ack packet
            ackHeader=(tcphdr*)malloc(STCP_MSS);
```

```
            stcp_network_recv(sd,ackHeader,STCP_MSS);
            if(ackHeader->th_flags!=TH_ACK){
                flag=1;
                goto label2;

            }
            // printf("ack:%u seq:%u
ini:%u\n",ntohl(ackHeader->th_ack),ntohl(ackHeader->th_seq),ctx->initial_sequence_num);
            label2:

            // set errno
            if(flag){
                printf("Some error\n");
                errno=ECONNREFUSED;
            }
        }
        else{
            printf("problem");
        }
        free(ackHeader);
        free(stcpHeader);
     }

    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 =;*/
    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)
{
    uint8_t OFF = 5;
    assert(ctx);
    assert(!ctx->done);
    // end point of window
    int seqIndex = ctx->initial_sequence_num+1;
    int finSent = 0;
    int finRecv = 0;
    int finAck = 0;
    int lastAck=seqIndex-1;
    int maxSeq=lastAck+3072;
    int wait=0;
    while (!ctx->done)
    {
        unsigned int event;

        /* see stcp_api.h or stcp_api.c for details of this function */
        /* XXX: you will need to change some of these arguments! */
        // In case packet is out of window bounds
        if(!wait)
            event = stcp_wait_for_event(sd,ANY_EVENT,NULL);
        else
            event = stcp_wait_for_event(sd,TIMEOUT | NETWORK_DATA |
APP_CLOSE_REQUESTED,NULL);

        /* check whether it was the network, app, or a close request */
        if (event & APP_DATA)
        {
            // printf("lastAck:%d seqIndex:%d maxSeq:%d\n",lastAck,seqIndex,maxSeq);
            if(!(seqIndex>=lastAck && seqIndex<maxSeq)){
                // printf("Out of sender window packet\n");
                wait=1;
            }else{
                // In case some part of data is outside window bound
                int dataWilling = MIN(STCP_MSS,maxSeq-seqIndex);
                char recvData[dataWilling];
```

```
            tcphdr dataHeader;
            dataHeader.th_seq = htonl(seqIndex);
            dataHeader.th_off = sizeof(tcphdr)/4;//sizeof(uint32_t);
            dataHeader.th_flags = 0;
            int dataRecv = stcp_app_recv(sd,recvData,dataWilling);

stcp_network_send(sd,&dataHeader,sizeof(dataHeader),recvData,dataRecv,NULL);
            seqIndex += dataRecv;
         }
      }else if(event & NETWORK_DATA){
         tcphdr *dataFromNetHeader;
         int maxPacketSize = STCP_MSS + sizeof(tcphdr);
         dataFromNetHeader = (tcphdr*)malloc(maxPacketSize);
         uint dataRecv = stcp_network_recv(sd,dataFromNetHeader,maxPacketSize);
         if(dataRecv==0){
            return ;
         }
         // proper endianess
         dataFromNetHeader->th_seq=ntohl(dataFromNetHeader->th_seq);
         dataFromNetHeader->th_win=ntohs(dataFromNetHeader->th_win);
         dataFromNetHeader->th_ack=ntohl(dataFromNetHeader->th_ack);
         // printf("%u %u\n",dataFromNetHeader->th_seq,dataFromNetHeader->th_ack);
         if(dataFromNetHeader->th_flags==TH_FIN){
            // TODO: handle fin+data packet
            finRecv=1;
            // data + FIN packet
            if(dataRecv>sizeof(tcphdr)){
               char dataFromNet[STCP_MSS];
               // get data portion
               strncpy(dataFromNet,
((char*)dataFromNetHeader)+TCP_DATA_START(dataFromNetHeader),
                  dataRecv-TCP_DATA_START(dataFromNetHeader));

               stcp_app_send(sd,dataFromNet,dataRecv-sizeof(tcphdr));
               tcphdr sendAckHeader;
               sendAckHeader.th_seq = htonl(dataFromNetHeader->th_seq+1);
               sendAckHeader.th_ack =
htonl(dataFromNetHeader->th_seq+dataRecv-sizeof(tcphdr));
               sendAckHeader.th_flags = TH_ACK;
               sendAckHeader.th_win = htons(3072);
               sendAckHeader.th_off = sizeof(tcphdr)/4;//sizeof(uint32_t);
               stcp_network_send(sd,&sendAckHeader,sizeof(sendAckHeader),NULL);
            }
            // without data
            tcphdr finAckPacket;
            finAckPacket.th_flags=TH_ACK;
```

```
            finAckPacket.th_seq=htonl(dataFromNetHeader->th_seq+1);
            finAckPacket.th_ack=htonl(dataFromNetHeader->th_seq+1);
            finAckPacket.th_off=OFF;

            // send acknowledgment
            stcp_network_send(sd,&finAckPacket,sizeof(finAckPacket),NULL);
            stcp_fin_received(sd);
            // If my fin acked then exit
            if(finAck){
               // close connection
               ctx->done=1;
            }
         }
         else if(dataFromNetHeader->th_flags==TH_ACK){
            // printf("Here\n");
            lastAck=dataFromNetHeader->th_ack;
            maxSeq = lastAck+dataFromNetHeader->th_win;
            wait=0;
            // If fin sent
            if(finSent){
               // If fin acked then quit else remember the ack
               if(((uint)seqIndex)==dataFromNetHeader->th_ack-1){
                  if(finRecv)
                     ctx->done=1;
                  else
                     finAck=1;
               }
            }
         }
         else{
            // printf("else\n");
            //data packet
            char dataFromNet[STCP_MSS];
            strncpy(dataFromNet,
((char*)dataFromNetHeader)+TCP_DATA_START(dataFromNetHeader),
               dataRecv-TCP_DATA_START(dataFromNetHeader));
            // send data to app
            stcp_app_send(sd,dataFromNet,dataRecv-sizeof(tcphdr));
            tcphdr sendAckHeader;
            sendAckHeader.th_seq = htonl(dataFromNetHeader->th_seq+1);
            sendAckHeader.th_ack =
htonl(dataFromNetHeader->th_seq+dataRecv-sizeof(tcphdr));
            sendAckHeader.th_flags = TH_ACK;
            sendAckHeader.th_win = htons(3072);
            sendAckHeader.th_off=OFF;
            // send ack
```

```
            stcp_network_send(sd,&sendAckHeader,sizeof(sendAckHeader),NULL);
        }
        free(dataFromNetHeader);
    }else if(event & APP_CLOSE_REQUESTED){
        // send fin packet
        tcphdr finPacket;
        finPacket.th_flags=TH_FIN;
        finPacket.th_seq = htonl(seqIndex);
        finPacket.th_off = OFF;
        stcp_network_send(sd,&finPacket,sizeof(finPacket),NULL);
        finSent = 1;
    }else if(event & TIMEOUT){
        printf("There shouldn't have been a timeout\n");
    }
  }
}


/*****************************************************************/
/* 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);
}
```