Ryerson University

Department of Electrical & Computer Engineering
Faculty of Engineering & Architectural Science

Course Title:	Computer Networks
Course Number:	COE 768
Semester/Year (e.g.F2016)	F2018

Instructor:	Bobby Ngok-Wah Ma

Assignment/Lab Number:	
Assignment/Lab Title:	Lab Project

Submission Date:	2018-11-15
Due Date:	2018-11-15

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Sahayathasan	Earvin	500709422	06	

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Introduction

The project covered was on a socket to socket connection thus the further in depth of socket programming and socket connection that is known the clearer the topic and project becomes. The definition of socket is when multiple software applications connect between one another through a point to point connection. Through this application a field of programming concepts can arise to develop this which was the foundation of socket programming. Socket programming is the coding aspect of how to send users information using different types of socket connections. A few types of connections learned were the TCP and UDP connections. However, in the lab conducted there was an UDP connection used between the sockets. The type of connection refers to the protocol or the rules that govern the movement of the information along the socket. The User Socket Protocol (UDP) is a communication protocol that uses low latency and low connection between application and internet so that the individual can send information through the socket without much effort. This type of connection is typical between application to application uses where the messages transferred are called datagrams and are used as the socket information that will be transferred. With this understanding on what the components and functions the project does the application performed will be more sensible.

The project conducted was to utilizing the fundamental concepts of computer networks in a real time application using socket programming. The project that was performed was meant to execute an application form which 2 client's networks are connected through a socket and using that connection send files and information towards each other across that line. The project that was done uses the socket programming concepts to allow these clients to send information by assigning each a title for their roles. In the case of this lab, the roles assigned were the 'server' and the 'client'. These two titles are given to one and the other person in the socket connection. The role identifies what type of action they can do but as a socket each connected user to the socket can attain and give information because sockets send information in both directions. The server and the client side both had the same functions and capabilities since this a level connection between 2 people but the titles just depict a direction on who is sending to who for the project's understanding. The files can be uploaded, downloaded, listed in a file directory and can change paths. These applications are the standard Protocol Data Units (PDU) which are the applications that will be used within the socket connection being used for the specific type of connection.

Description

The four general tasks that were meant to be demonstrated for the project was the registration which means registering the files into the directory, the download which receives files from the directory, the listing of online registration content which is the list of all files within the directory and finally the content deregistration which is the delete option for unused files in the directory. These tasks are also referred to as PDU's which are the functions of the socket connection that was created. Some other PDU's that were not explicitly shown used are the background PDU's which confirm the connection properly sends the correct material. The PDU's not mentioned were the Search used to find contents in the directory, Content Data which confirms the data in the entry exists, Acknowledgement which confirms to the initial user

if the action performed was approved and next performance will continue and finally, Error which triggers if any errors occur at any point and time. Theses PDU's and the socket connection initially takes a file from the client and then uploads it to the server which has name types associated but then returns signals from the server from the acknowledgement or the error PDU. Progress may continue only if no errors and acknowledgement has been received which shows the utility of this PDU.

Client

There are 2 members in this socket connection and the first one discussed will be the client. The client was asked to perform several tasks to show each PDU for the user functions correctly. The client's main objective was to upload a file through the socket connection to the server than after receive a file by downloading it from the socket connection. The client was only able to perform this by connecting directly to the socket of the server directory. The server directory also referred to as the 'sd' is the location of the socket connection between the server and client for the project. The program checks if the action has successfully connected by using the PDUS from the echo functions to get an acknowledgement from the server side. These PDU's that the client used were done as shown:

1) R – Content Registration

   The content registration is a request sent from the user which can be called the 'R' PDU. This PDU basically takes a file from the user and registers it into the server directory where the files that are listed will be stored and accessible later. The user waits for the acknowledgement before proceeding with the transfer. When acknowledged the user will send the content data PDU notifying the directory that the file is ready to send. The filename will be checked to see if it has prior usage and if not then continues otherwise the error PDU will trigger stopping this transfer until the issue found is resolved.

2) D – Download

   The download option can be requested by referring to the 'D' PDU. When the user sends the request the server end will respond back with an acknowledgement by sending the acknowledge PDU to requesting user. Once sent the user will be asked for the filename. This portion takes the file name and uses the Search PDU to look through the directory and if found fills in the Content Data PDU to notify its existence. If the content data is null meaning no found results than the error PDU will occur signaling that the request has had an error and the transfer stops. Once the transaction completes that specific request ends and waits for another one.

3) T – Content Deregistration

   The content deregistration takes a unit in the server directory and removes it from the directory. This request is referred to as the 'T' PDU. The user essentially takes a filename

and determines if the contents exist in the library using the search and content data PDU. If successful an acknowledgement of removal will be sent using the acknowledgment PDU otherwise if an error with incapability of deletion or non-existent filename occurs than the error PDU trigger will respond accordingly to the user.

4) O - List of Online Registration Content

Another PDU used was the listing of the online registration content where the PDU term is referred to as the 'O' PDU. The list PDU returns a list of the entries in the server directory at that point in time. The program calls from the search PDU to search the data in the server directory for what is needed. The content data PDU is activated once a match has occurred and the requested content was found. The list may prompt errors when a filename was searched but it does not exist thus Null Error which triggers the error PDU again.

Server

The other member of the socket connection was the server where the server also uses the same functionalities as the client. This is only occurring because the point of this connection is for any two users to connect to one another across a LAN connection to receive information from another. The server side can also be seen as another client side because likewise this client has to send files to the client user through the server directory which is the point of intersection between these users. The PDU's likewise will perform the same and will have the same background functionalities. However, for acknowledgement responses the client will now send this information back to the server side to confirm if all actions undergone are capable or an error has occurred. Once completed than the task has been completed and will discontinue execution until called again. The PDU descriptions can be used from the clients since the application does not change depending on the user.

Observations and Analysis

On the client side:
The client will send a file to the server to upload and then will upload another to confirm for multiple entries in the directory.
Upload/ Register Client Side: Client_Upload.txt and Client_Upload2.txt

On the server side:

The flag is the indicator for which directory to take from because depending on which the user will take from the root or from the server's directory: flag.txt

These are the files that the server can put the uploaded/registered files into so that it will be in the server directory: Server_Download.txt, Server_Download2.txt and Server_Download3.txt

These are the files that have been modified and are accessed by the client from the server from the download PDU: Server_Upload.txt and Server_Upload2.txt

All the functions in the program ran smoothly and efficiently. The server and client both run all the PDU functions effectively and clearly. The server had been tested for multiple uploads, downloads, listing of the directories and deregistration. The functionalities all worked cohesively however when thrown an error when told to search for a non-existing file the correct PDU prompt appeared in the notification of an error message saying that the file asked does not exist and continues to run when unable to find. The functions likewise performed the same when testing on the client side as well. This was in regards to the being able to acknowledge all the returns from the server and being able to respond back accordingly. The messages were never lost unless it was manually deregistered from the server and client sides within the socket. The errors were also working accordingly for non-existing files checking and deregistering and it worked correctly as well. Overall performance, of the lab was complete and successful.

Conclusion

The performed project successfully executed all the PDUs correctly and successfully transferred files from the clients to another which was the objective of the lab. The UDP allowed the transmission between the users to be very simple and effective because using this protocol allowed each user to access what they needed and confirm with their counterparts when doing so through the acknowledgement PDU. The lab was designed to further understand the uses of transmission in an applicable use to deeper understand socket programming which truly resulted in a greater learning experience.

Through this lab many other applications can be seen using this and are much clearer to conceptualize when attempting to tackle these problems later on. Such as server control units or cloud computing where users store files into the cloud and get immediate responses and transmissions back from them when needed to be accessed. These are just some examples of how applicable networks and specifically socket programming can be used in real time and this lab clearly showed that as well.

Appendix

```
#include <sys/stat.h>

#define SERVER_TCP_PORT 3000    /* well-known port */
#define BUFLEN      256 /* buffer length */
#define LENGTH 512
#define dataSize 100

int echod(int);
void reaper(int);

//creation of pdu struct for use in echod()

struct pdu_header
            {
                    char type;
                    unsigned int len;
                    char data[dataSize];
            }server_header, receive_PDU, transfer_PDU, path;

int main(int argc, char **argv)
{
        int     sd, new_sd, client_len, port;
    struct  sockaddr_in server, client;

    switch(argc){
    case 1:
        port = SERVER_TCP_PORT;
        break;
    case 2:
        port = atoi(argv[1]);
        break;
        default:
            fprintf(stderr, "Usage: %d [port]\n", argv[0]);
            exit(1);
        }

        /* Create a stream socket   */
        if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
            fprintf(stderr, "Can't create a socket\n");
            exit(1);
        }

        /* Bind an address to the socket    */
        bzero((char *)&server, sizeof(struct sockaddr_in));
        server.sin_family = AF_INET;
        server.sin_port = htons(port);
        server.sin_addr.s_addr = htonl(INADDR_ANY);
        if (bind(sd, (struct sockaddr *)&server, sizeof(server)) == -1){
            fprintf(stderr, "Can't bind name to socket\n");
            exit(1);
        }

        /* queue up to 5 connect requests  */
        listen(sd, 5);

        (void) signal(SIGCHLD, reaper);

        while(1) {
          client_len = sizeof(client);
          new_sd = accept(sd, (struct sockaddr *)&client, &client_len);
          if(new_sd < 0){
            fprintf(stderr, "Can't accept client \n");
```

```
        exit(1);
      }
      switch (fork()){
      case 0:       /* child */
        (void) close(sd);
        exit(echod(new_sd));
*     default:      /* parent */
        (void) close(new_sd);
        break;
      case -1:
        fprintf(stderr, "fork: error\n");
      }
    }
}

/*  echod program   */
int echod(int sd)
{
    char file_name[256];
    int n, x;
    DIR *directory;
    struct dirent *ent;


    char flagArray[1];
    FILE *flag = fopen("flag.txt","r");
    fread(flagArray,1,256,flag);
    fclose(flag);

    //Receive PDU from client side

    x= read(sd,&server_header,BUFLEN);
    server_header.data[x-1] = '\0';

//Switch statement to determine which case to execute

switch(server_header.type){


//Upload Case

    case 'U':

//Send ready signal to client

        transfer_PDU.type ='R';
        write(sd,&transfer_PDU,BUFLEN);

//Receive filename from client

        n = read(sd,&receive_PDU,BUFLEN);

//Send ready signal to client

        printf("Filename: %s\n",server_header.data);
        printf("Receiving %s\n", receive_PDU.data);
        transfer_PDU.type = 'R';
        write(sd,&transfer_PDU, BUFLEN);

       //Check flag
        fread(flagArray,1,256,flag);
```

```
        flagArray[1] = '\0';

      //Determine appropriate directory

        if(strcmp(flagArray,"1") == 0){
            printf("Folder: test\n");
            strcpy(file_name,"./test/");
            strcat(file_name, server_header.data);
            strcat(file_name,".txt");
            FILE *fl = fopen(file_name,"w");
            fprintf(fl,"%s",receive_PDU.data);

        }

        else{
            printf("Folder root\n");
            strcpy(file_name, server_header.data);
            strcat(file_name,".txt");
            FILE *fl = fopen(file_name,"w");
            fprintf(fl,"%s",receive_PDU.data);
        }

    break;

//Download case
    case 'D' :

       server_header.data[server_header.len-1] = '\0';
        fread(flagArray,1,256,flag);

        flagArray[1] = '\0';

//Check flag to see which folder to look inside

        //Test folder contains two files
        if(strcmp(flagArray,"1")== 0){

            if(strcmp(server_header.data,"joker") == 0){
                FILE *fa = fopen("./test/joker.txt","r");
                n = fread(transfer_PDU.data,1,256,fa);
            }
            else if(strcmp(server_header.data,"batman") == 0){
                FILE *fl = fopen("./test/batman.txt","r");
                n = fread(transfer_PDU.data,1,256,fl);
            }

             else{
                transfer_PDU.type = 'E';
                strcpy(transfer_PDU.data,"File not found!");
                transfer_PDU.len = strlen(transfer_PDU.data);
                write(sd,&transfer_PDU,BUFLEN);

            }
        }
        //root folder contains name.txt and super.txt
        else{
            //strcpy(file_name, server_header.data);
            //strcat(file_name,".txt");
            //FILE *f1 = fopen(file_name,"r");
            if(strcmp(server_header.data,"name")==0){
                FILE *fl = fopen("name.txt","r");
                 n = fread(transfer_PDU.data,1,256,fl);
                }
```

```
            else if (strcmp(server_header.data,"super")==0){
                FILE *fl = fopen("super.txt","r");
                n = fread(transfer_PDU.data,1,256,fl);
                }
            else{
                transfer_PDU.type = 'E';
                strcpy(transfer_PDU.data,"File not found!");
                transfer_PDU.len = strlen(transfer_PDU.data);
                write(sd,&transfer_PDU,BUFLEN);
                }

        }

       //Send file transfer signal to client
        if (n>0){

            transfer_PDU.len = strlen(transfer_PDU.data);
            int a = transfer_PDU.len;
            transfer_PDU.data[a-1]='\0';
            transfer_PDU.type = 'F';
            write(sd, &transfer_PDU, BUFLEN);
            printf("File transfer completed);
            }
    break;

//listing the files in current folder
    case 'L' :
        printf("Directory List: \n");
        server_header.data[server_header.len-1] = '\0';
        //get the directory info from test folder
        if(strcmp(server_header.data,"test") == 0){


            directory = opendir("./test");
            printf("%s\n",directory);
        }
        //get the directory info from root folder
        else if(strcmp(server_header.data,"root")==0){
            directory = opendir("./");
            printf("%s\n",directory);
        }
        else{
            transfer_PDU.type = 'E';
            strcpy(transfer_PDU.data,"Directory Could Not Be Found (0)");
            write(sd,&transfer_PDU,BUFLEN);
        }

        if(directory){

            while ((ent = readdir(directory))!= NULL){
                strcat(transfer_PDU.data,ent->d_name);
                strcat(transfer_PDU.data,"\n");
            }
            transfer_PDU.type ='I';
                printf("Sending \n");
                printf("Data: %s\n",transfer_PDU.data);
                printf("Type: %c\n",transfer_PDU.type);
                write(sd,&transfer_PDU, BUFLEN);
            closedir (directory);
        }
        // report error ,if the path name does not match
        else
        {
            transfer_PDU.type = 'E';
```

```
            strcpy(transfer_PDU.data,"Directory Could Not Be Found");
            write(sd,&transfer_PDU,BUFLEN);
        }
    break;

    //Path change case

 case 'P':

       FILE *flag = fopen("flag.txt","w");

        server_header.data[server_header.len-1] = '\0';
        if(strcmp(server_header.data,"test")==0){
            transfer_PDU.type = 'R';
            fprintf(flag,"%s","1");
        }
        else if(strcmp(server_header.data,"root")==0){
            transfer_PDU.type = 'R';
            fprintf(flag,"%s","0");
        }
        else{

            transfer_PDU.type = 'E';
            strcpy(transfer_PDU.data,"Directory Could Not Be Found");
            transfer_PDU.len = strlen(transfer_PDU.data);
            write(sd,&transfer_PDU,BUFLEN);

        }
        write(sd,&transfer_PDU,BUFLEN);

    break;
}
    close(sd);
    return(0);
}

/*  reaper      */
void    reaper(int sig)
{
    int status;
    while(wait3(&status, WNOHANG, (struct rusage *)0) >= 0);
}
```

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/signal.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include <dirent.h>
#include <unistd.h>
#include <sys/stat.h>
#include <netdb.h>

#define SERVER_TCP_PORT 3000    /* well-known port */
#define BUFLEN      256 /* buffer length */
#define LENGTH 512
#define dataSize 100
struct pdu_header
            {
                    char type;
                    unsigned int len;
                    char data[dataSize];
            }server_header, transfer_PDU, receive_PDU, name, upload;

int main(int argc, char **argv)
{
    int     n, i, bytes_to_read;
    int     sd, port;
    struct  hostent     *hp;
    struct  sockaddr_in server;
        struct stat fstat;
    char    type,*host, *bp, rbuf[BUFLEN], sbuf[BUFLEN],wbuf[BUFLEN];
        char f_name[256];

    switch(argc){
    case 2:
        host = argv[1];
        port = SERVER_TCP_PORT;
        break;
    case 3:
        host = argv[1];
        port = atoi(argv[2]);
        break;
    default:
        fprintf(stderr, "Usage: %s host [port]\n", argv[0]);
        exit(1);
    }

    /* Create a stream socket    */
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        fprintf(stderr, "Can't creat a socket\n");
        exit(1);
    }

    bzero((char *)&server, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_port = htons(port);
    if (hp = gethostbyname(host))
       bcopy(hp->h_addr, (char *)&server.sin_addr, hp->h_length);
```

```
        else if ( inet_aton(host, (struct in_addr *) &server.sin_addr) ){
          fprintf(stderr, "Can't get server's address\n");
          exit(1);
        }

        /* Connecting to the server */
        if (connect(sd, (struct sockaddr *)&server, sizeof(server)) == -1){
          fprintf(stderr, "Can't connect \n");
          exit(1);
        }

            int length;
           //Menu - Give options to user and wait for user input
            printf("What Would You Like to Do Today? \n");
            printf("Enter D for Download \n");
            printf("Enter U for Upload \n");
            printf("Enter L for a list of files within a directory\n");
            printf("Enter P to change the path\n");

            scanf("%c",&type);

            //Download Case

            if(type == 'D'){
                printf("Enter the name of the file you wish to download (without file extension): \n");
                n= read(0,transfer_PDU.data,256);

   transfer_PDU.len = strlen(transfer_PDU.data);
                transfer_PDU.type = 'D';
                strcpy(f_name,transfer_PDU.data);
                strcat(f_name,".txt");

write(sd,&transfer_PDU,BUFLEN);

n= read(sd,&receive_PDU,BUFLEN);
            //received pdu is not type F,there is a transmission error
            if(receive_PDU.type !='F'){
                printf("Error Occured.File not Received. \n");

                }
            else {
   FILE *fp = fopen(f_name, "w+");
                fprintf(fp,"%s",receive_PDU.data);
                printf("File Received.\n");
                fclose(fp);
            }
        }

        //Upload Case

        else if(type == 'U'){

            //Prompt user for name of file to upload

            printf("Enter the name of the file you wish to upload (without file extension): ");
            n = read(0, &name.data, BUFLEN);
            name.data[n-1] = '\0';
            name.len = strlen(name.data);
            name.type= 'U';
            printf("Name of file to upload: %s\n", name.data);
            write(sd,&name,BUFLEN);
```

```
            n = read(sd,&server_header,BUFLEN);

    if (server_header.type == 'R'){
                upload.type = 'F';
         printf("Sending %s.\n",name.data);
                if(strcmp(name.data,"happy")==0){
                    FILE *fp = fopen("happy.txt","r");
                    n = fread(upload.data,1,256,fp);
                    write(sd,&upload,BUFLEN);//printf("Sending %s\n",name.data);
                }
                else if(strcmp(name.data,"revolver")==0){
                    FILE *fp = fopen("revolver.txt","r");
                    n = fread(upload.data,1,256,fp);
                    write(sd,&upload,BUFLEN);//printf("Sending \n",name.data);
                }
                else{

                    write(sd,"error\n",6);
                    perror("Cannot open file");
                }


            }

        }
        //Path Change

        else if(type == 'P'){
            //asking for the folder name and send request to server
            transfer_PDU.type = 'P';
            printf("Enter folder name you want to go(root/test): \n");
            n = read(0,&transfer_PDU.data,BUFLEN);
            transfer_PDU.len = strlen(transfer_PDU.data);
            write(sd,&transfer_PDU,BUFLEN);
            n= read(sd,&receive_PDU,BUFLEN);
            //if server replied with R type pdu, the path change is successful
            if(receive_PDU.type != 'R'){
                printf("Server not ready\n");
            }
            else{
                printf("path changed\n");
            }

        }
// Directory File List

        else if(type == 'L'){
            transfer_PDU.type = 'L';
            printf("Enter Directory(root/test): \n");
            n = read(0,&transfer_PDU.data,BUFLEN);
            transfer_PDU.len = strlen(transfer_PDU.data);
            write(sd,&transfer_PDU,BUFLEN);

            n = read(sd,&receive_PDU,BUFLEN);

if(receive_PDU.type == 'I'){
                printf("List of directories: \n");
                printf("%s\n",receive_PDU.data);
            }
            else if(receive_PDU.type == 'E'){
                printf("%c\n",receive_PDU.type);
                printf("No such directory matches");
```

```
            }
            else{
                printf("%c",receive_PDU.type);
                printf("Error\n");
            }
        }

    close(sd);
    return(0);
}
```