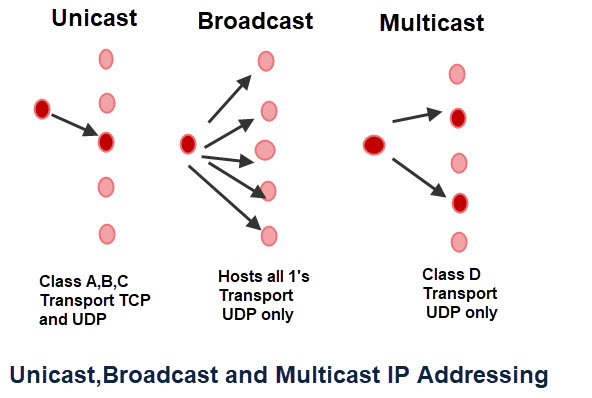
# Internet Radio: Multicasting Multimedia over IP

# Project Report

## Introduction

The purpose of this project was to explore the IP multicast and the nature of multimedia traffic. We developed an internet TV/radio. We used our experience of sending data over a TCP connection, sending multimedia over a UDP connection, and sending data in a structured manner from the previous labs. The Any Source Multicast (ASM) model was used for multicasting.



## Ideas for further development

This project can potentially be used for internet radio and live streaming of videos. Currently, this code is meant only for Linux systems but modifications in socket programming can make it viable for the windows platform too.

## Implementation and design

The code basically follows the default design. It uses multiple processes and not multi-threading. Termination of a station is done by killing the process.

The application contains a menu which has options for Pause, Reset, Change of Station and Close.

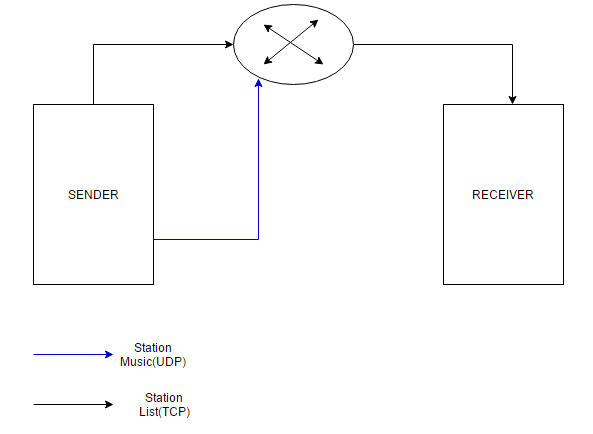
The flow of events is as follows:

* Client connects to the server
* Server sends channel information
* Client connects to the desired channel.
* Client starts receiving packets in a buffer.

## System Design

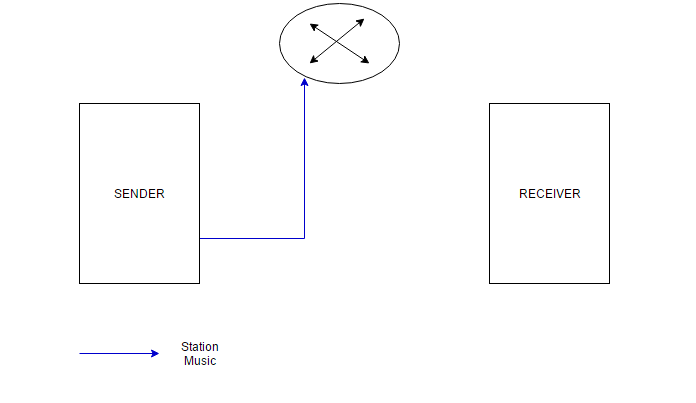
### Phase 1

Sender sends TCP packet to receiver with station information.



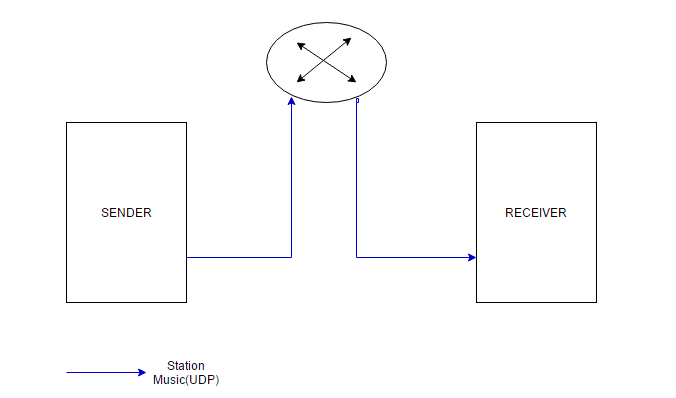
### Phase 2

Receiver selects preferred channel.



### Phase 3

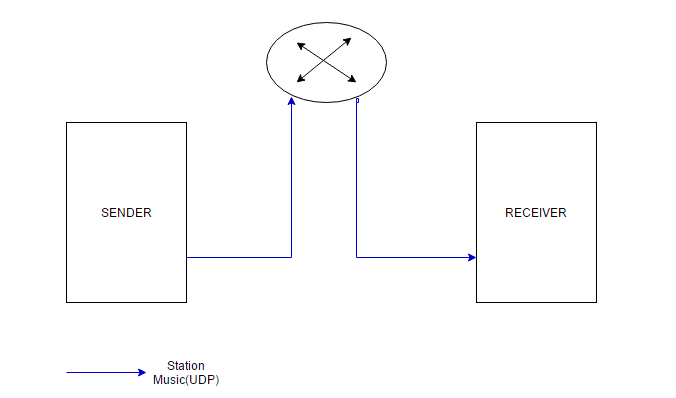
Receiver starts receiving the station music.



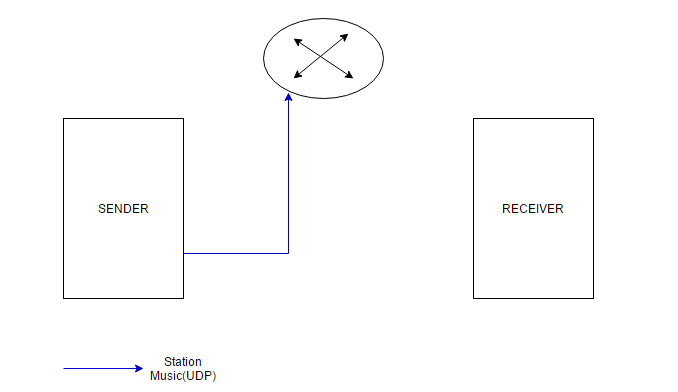
### Phase 4

(Reset, Pause, Change Station, Close)

#### Reset

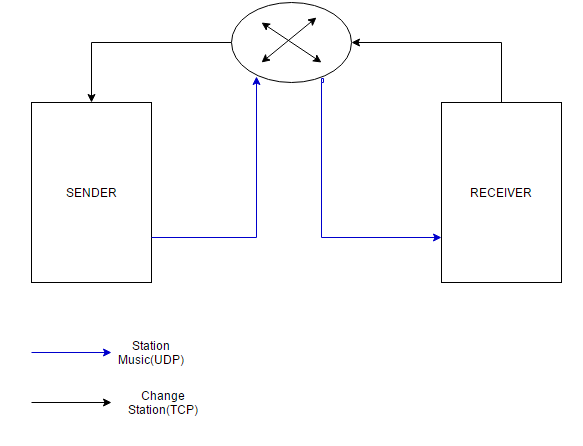


#### Pause

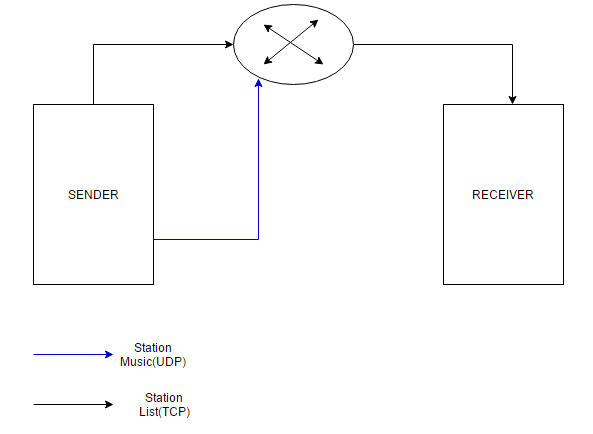


#### Change Station

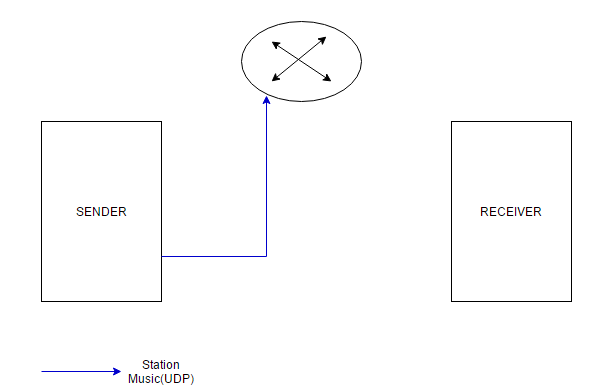
* Receiver requests for a change in station by sending a TCP packet.



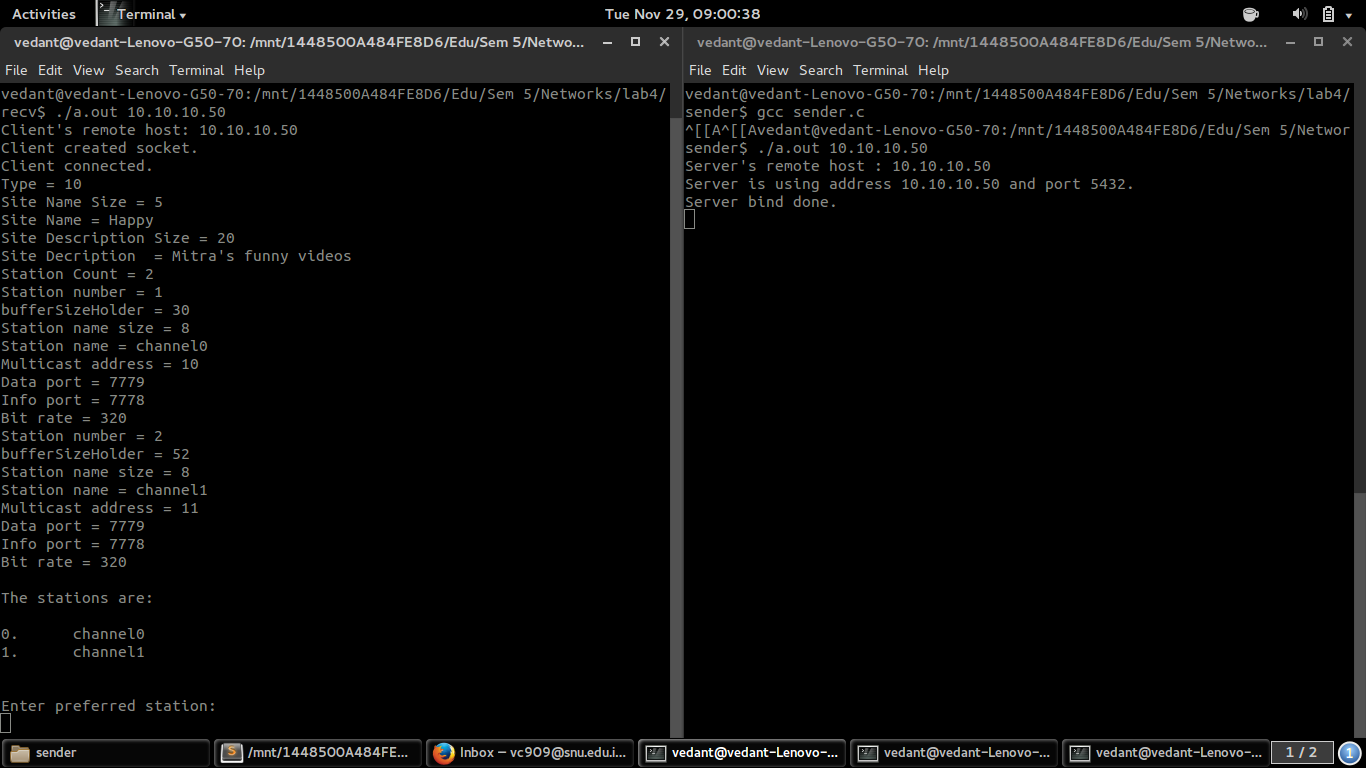
* Sender sends a TCP packet with the station list.

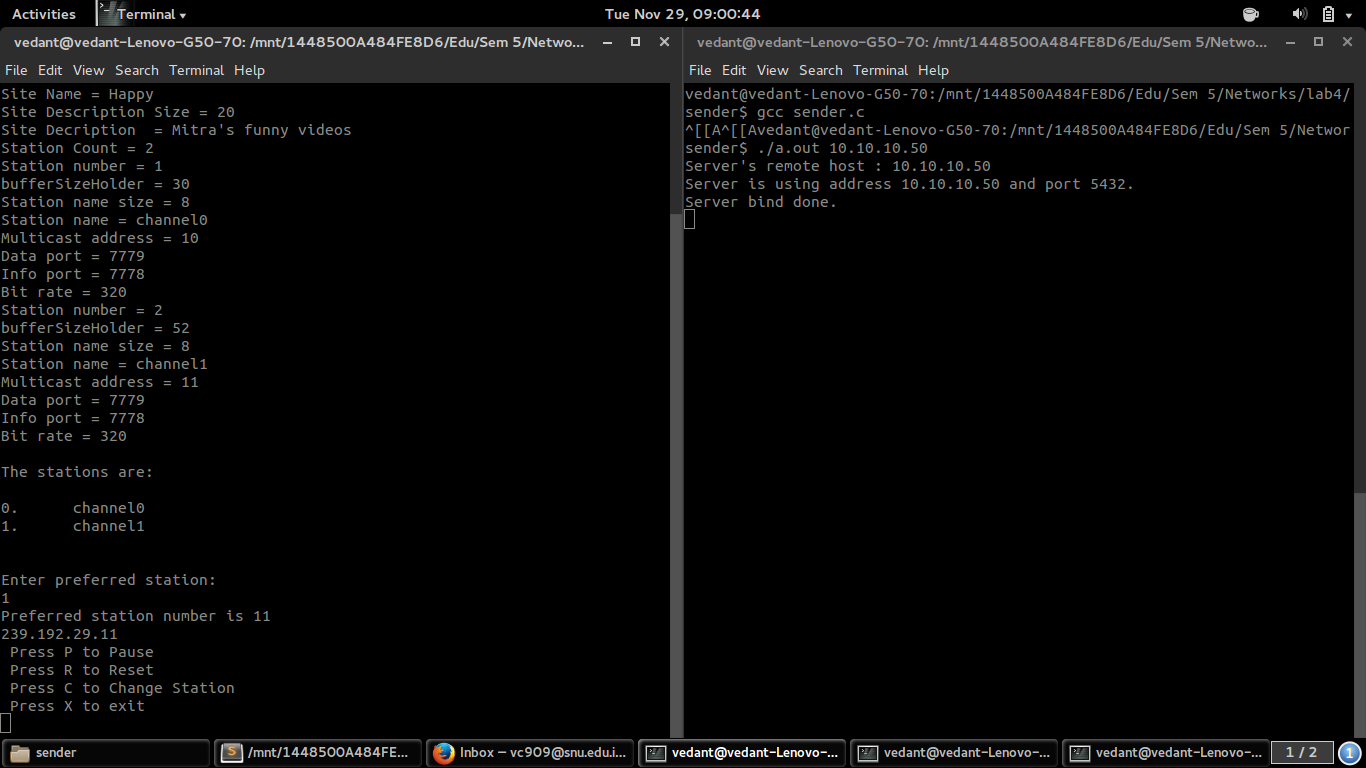


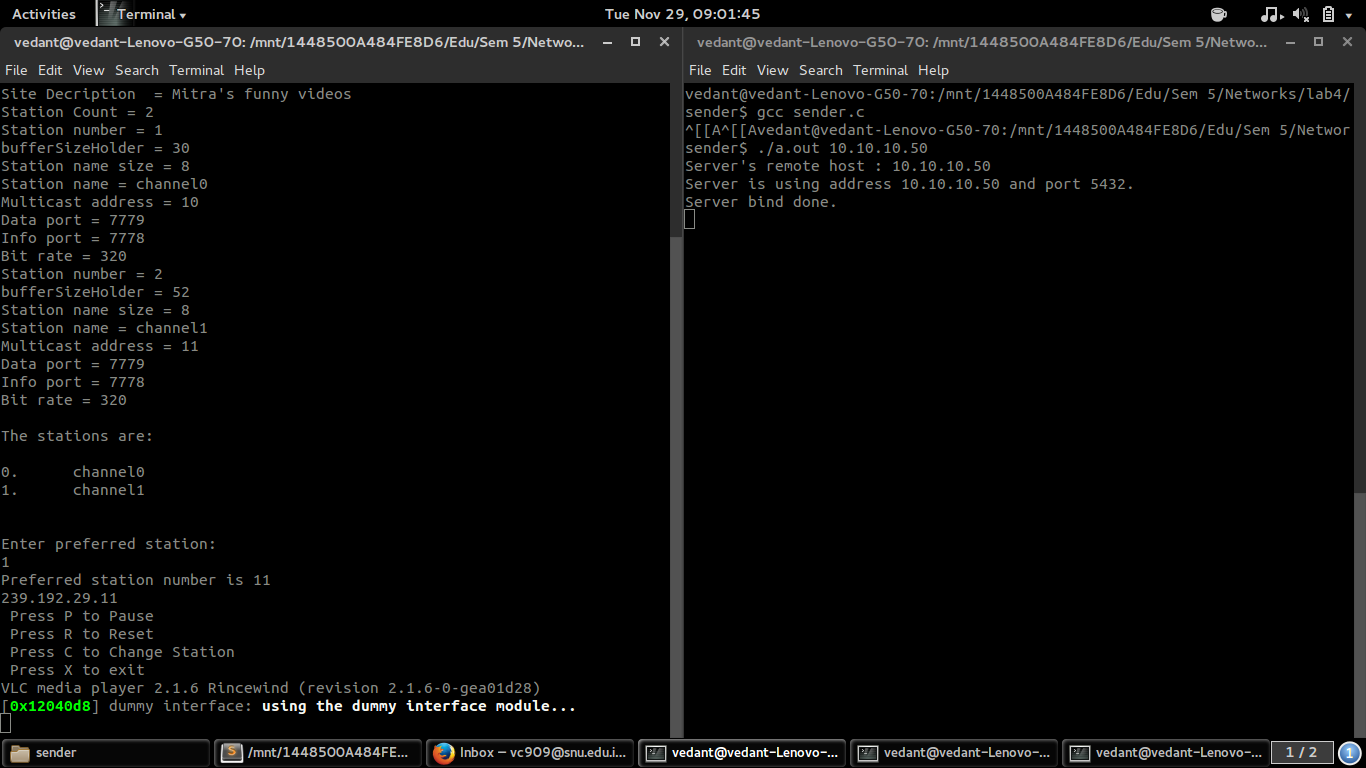
#### Close Station

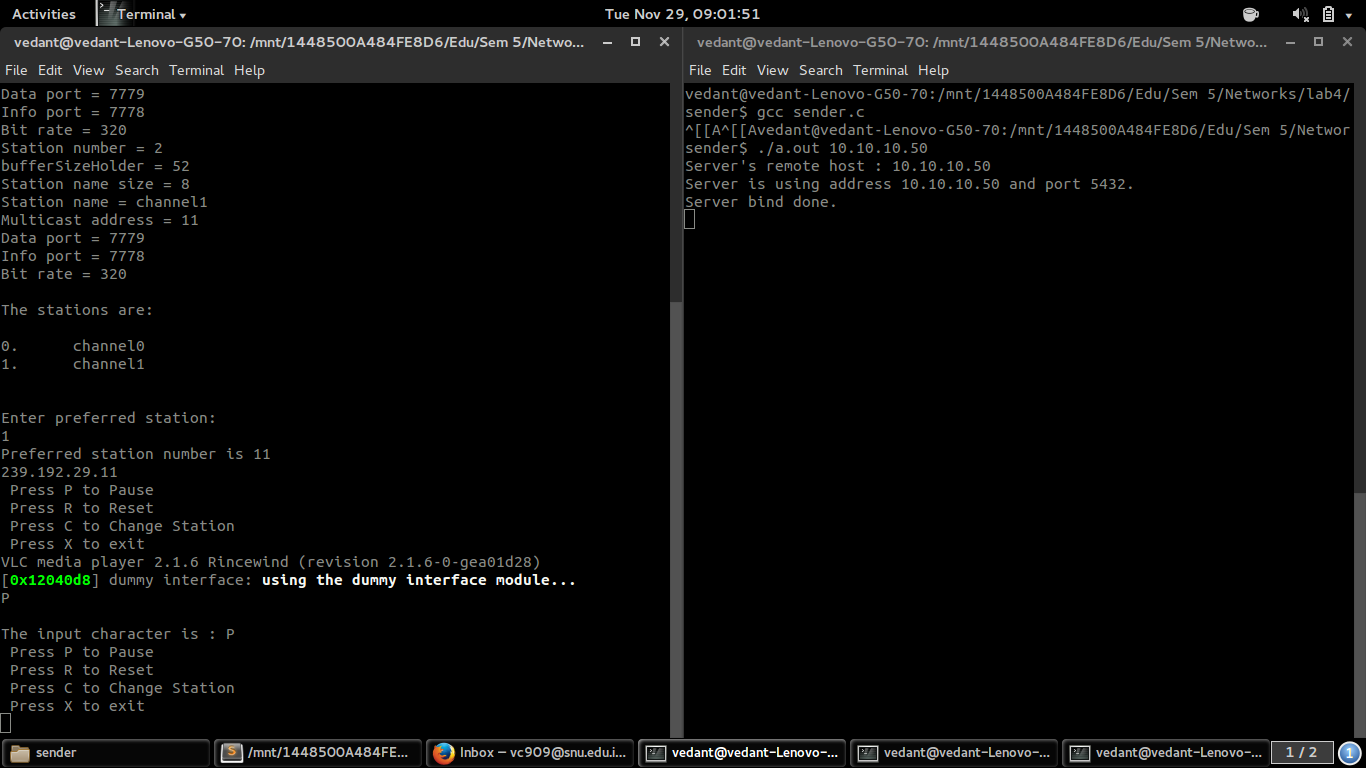


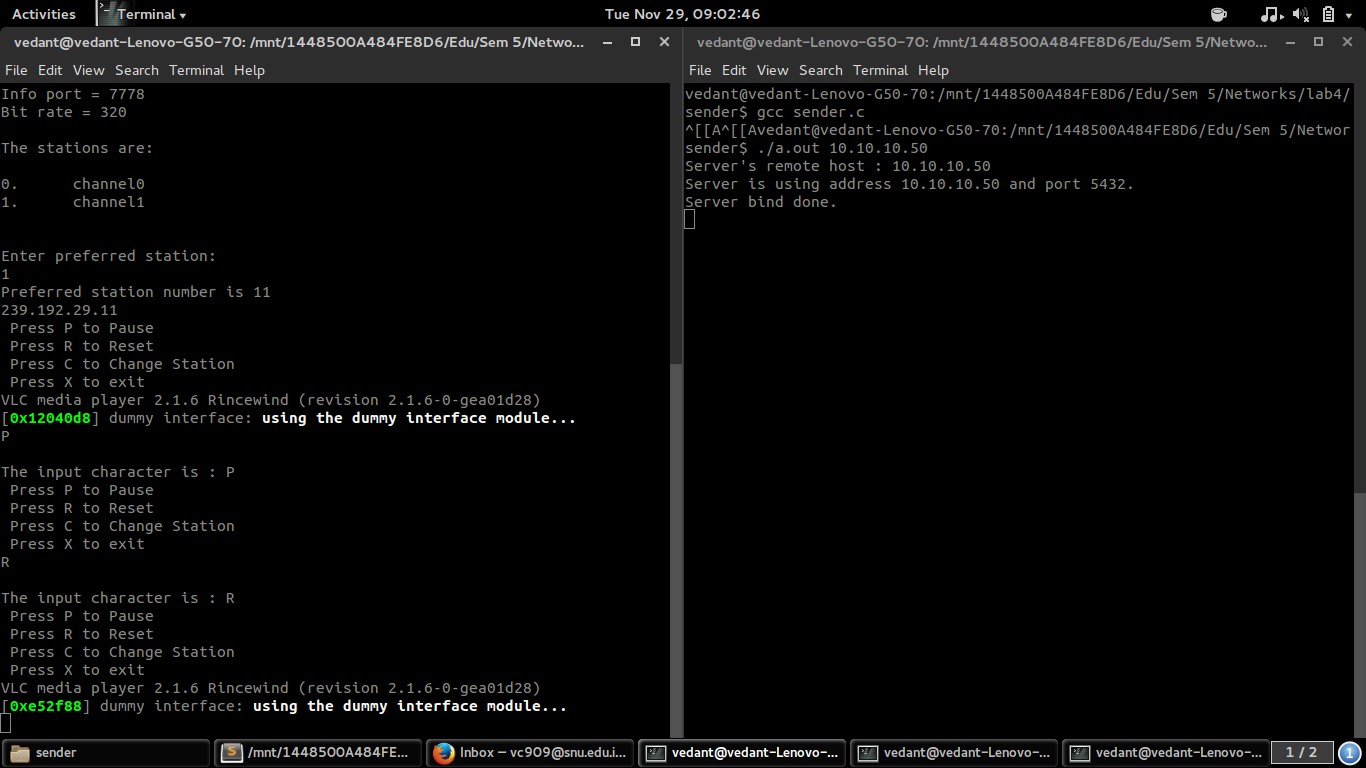
## Screenshots

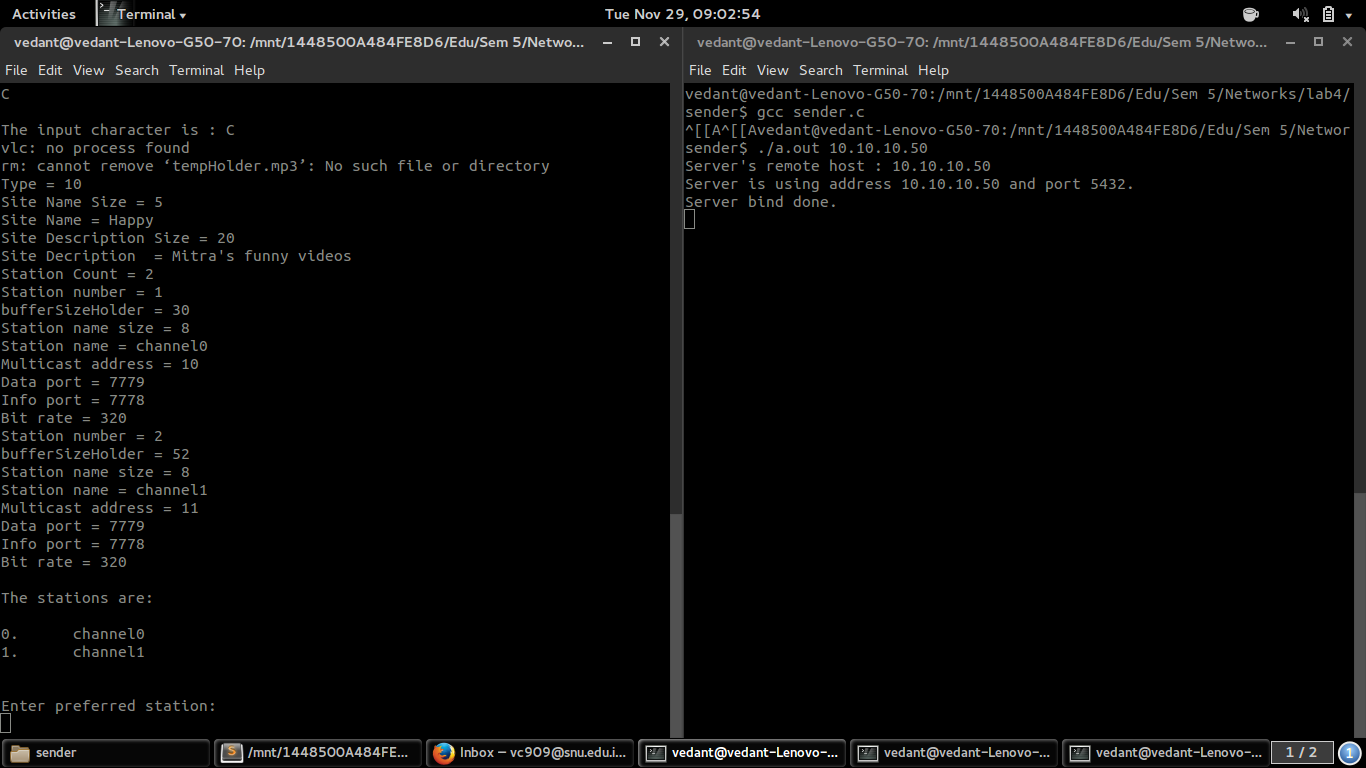


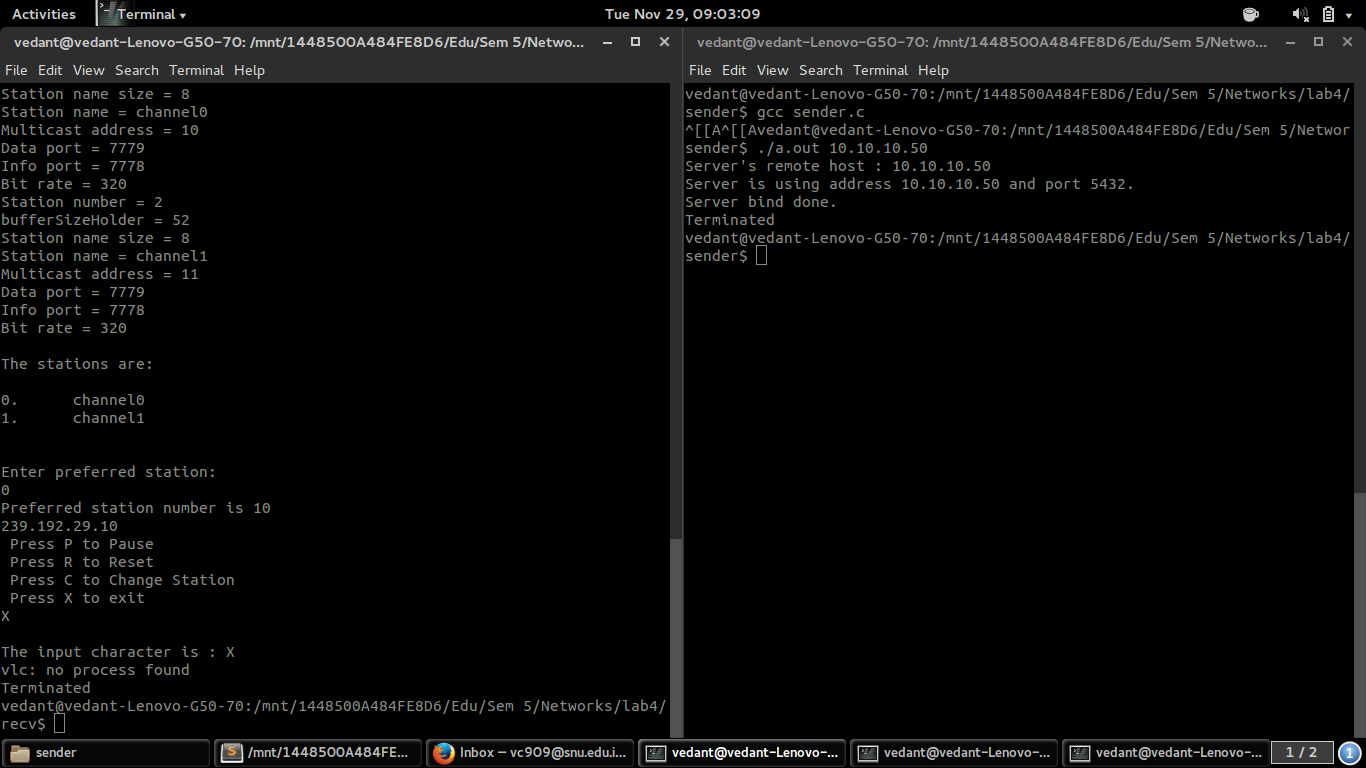












## Break up of individual contribution

**Team Number: 29**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Team Member Name** | **Contribution Detail** |
| 1 | Prasanna Natarajan | Client and Server code. Content management. |
| 2 | Siddharth Mitra | Report |
| 3 | Sridhar Ramanujam | Report |
| 4 | Vedant Chakravarthy | Client and Server code. Content management. |

## Code:

### Sender Side:

/\* CSD 304 Computer Networks, Fall 2016

multicast receiver

Team: Prasanna Natarajan

Siddhart Mitra

Sridhar Renga Ramanujam

Vedant Chakravarthy

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <net/if.h>

#include <netdb.h>

#include <sys/ioctl.h>

#include <signal.h>

#define TCP\_PORT 5432

#define MC\_PORT 7779

#define BUF\_SIZE 64480

#define DEBUG 0

#define DEBUG2 0

**typedef** **struct** station\_info\_request {

**uint8\_t** type;

}

**station\_info\_request\_t**;

**typedef** **struct** song\_info {

**uint8\_t** type;

**uint8\_t** song\_name\_size;

**char** song\_name;

**uint16\_t** remaining\_time\_in\_sec;

**uint8\_t** next\_song\_name\_size;

**char** next\_song\_name;

}

**song\_info\_t**;

/\*function prototypes\*/

**void** **Serialize\_station\_info\_req**(**station\_info\_request\_t** \* buf, **char** \* x);

/\*Helper Functions\*/

**void** **Serialize\_station\_info\_req**(**station\_info\_request\_t** \* buf, **char** \* x) {

**int** i;

**char** buffer[**1024**];

memcpy(buffer, & (buf -> type), **1**);

**if** (DEBUG == **1**) {

puts(">>Print memcpy buffer");

**for** (i = **0**; i < **1**; i++) {

printf("%d", buffer[i]);

}

}

memcpy(x, buffer, **sizeof**(buffer));

**if** (DEBUG == **1**)

puts("End Serialize\_station\_info\_req");

}

**int** **main**(**int** argc, **char** \* argv[]) {

**int** s, s\_tcp; /\* socket descriptor \*/

**struct** sockaddr\_in sin, sin\_tcp; /\* socket struct \*/

**char** \* if\_name; /\* name of interface \*/

**struct** ifreq ifr; /\* interface struct \*/

**char** buf[BUF\_SIZE];

**int** len;

/\* Multicast specific \*/

// char \* mcast\_addr; /\* multicast address \*/

**struct** ip\_mreq mcast\_req; /\* multicast join struct \*/

**struct** sockaddr\_in mcast\_saddr; /\* multicast sender\*/

**socklen\_t** mcast\_saddr\_len;

/\*TCP specific\*/

**char** \* tcp\_addr; /\*tcp ip address for control codes\*/

**struct** hostent \* hp;

/\* Add code to take port number from user \*/

**if** ((argc == **2**) || (argc == **3**)) {

tcp\_addr = argv[**1**];

}

**else** {

fprintf(stderr, "usage:(sudo) receiver tcp\_address [interface\_name (optional)]**\n**");

exit(**1**);

}

**if** (argc == **3**) {

if\_name = argv[**2**];

}

**else**

if\_name = "wlan0";

/\* translate host name into peer's IP address \*/

hp = gethostbyname(tcp\_addr);

**if** (!hp) {

fprintf(stderr, "simplex-talk: unknown host: %s**\n**", tcp\_addr);

exit(**1**);

}

**else**

printf("Client's remote host: %s**\n**", argv[**1**]);

/\* build address data structure \*/

bzero((**char** \* ) &sin\_tcp, **sizeof**(sin\_tcp));

sin\_tcp.sin\_family = AF\_INET;

bcopy(hp -> h\_addr, (**char** \* ) & sin\_tcp.sin\_addr, hp -> h\_length);

sin\_tcp.sin\_port = htons(TCP\_PORT);

/\* active open \*/

**if** ((s\_tcp = socket(PF\_INET, SOCK\_STREAM, **0**)) < **0**) {

perror("simplex-talk: socket");

exit(**1**);

}

**else**

printf("Client created socket.**\n**");

**if** (connect(s\_tcp, (**struct** sockaddr \*)&sin\_tcp, **sizeof**(sin\_tcp)) < **0**) {

perror("simplex-talk: connect");

close(s);

exit(**1**);

}

**else**

printf("Client connected.**\n**");

// puts("I beg of you, mister gobolan");

**station\_info\_request\_t** \* req;

req = malloc(**sizeof**(**struct** station\_info\_request) \* **sizeof**(**char**));

req -> type = **1**;

// puts("Creating station\_info\_request\_t");

**char** buf\_tosend[**1024**];

**char** receivedBuffer[**20000**];

**uint8\_t** decoded\_type;

**uint8\_t** decoded\_site\_name\_size;

**char** \* decoded\_site\_name;

**uint8\_t** decoded\_site\_desc\_size;

**char** \* decoded\_site\_desc;

**uint8\_t** decoded\_station\_count;

**int** bufferSizeHolder = **0**;

**uint8\_t** \* decoded\_station\_name\_size;

**char** \* \* decoded\_station\_name;

**uint32\_t** \* decoded\_multicast\_address;

**uint16\_t** \* decoded\_data\_port;

**uint16\_t** \* decoded\_info\_port;

**uint32\_t** \* decoded\_bit\_rate;

**char** buffer\_recv[BUF\_SIZE];

**int** tempx;

**char** \*default\_mcast\_addr;

default\_mcast\_addr = (**char**\*)malloc(**sizeof**(**char**)\***30**);

**int** preferred\_station\_number;

**pid\_t** forkResult;

**int** file\_counter = -**1**;

**int** scanfCounter = -**1**;

**while**(**1**)

{

CHANGE:file\_counter++;

// puts("before send loop starts");

bufferSizeHolder = **0**;

Serialize\_station\_info\_req(req, buf\_tosend);

send(s\_tcp, buf\_tosend, **1024**, **0**);

**if**(DEBUG == **1**)

puts("before recv");

recv(s\_tcp, receivedBuffer, **sizeof**(receivedBuffer), **0**);

**if**(DEBUG == **1**)

puts("After recv");

/\*

1) Decode the site\_info struct and put it into correponding variables

uint8\_t type;

uint8\_t site\_name\_size;

char \* site\_name;

uint8\_t site\_desc\_size;

char \* site\_desc;

uint8\_t station\_count;\*/

//Assuming everything is in a thing called char\* receivedBuffer

memcpy( & decoded\_type, receivedBuffer + bufferSizeHolder, **1**);

bufferSizeHolder += **1**;

printf("Type = %d**\n**",decoded\_type );

memcpy( & decoded\_site\_name\_size, receivedBuffer + bufferSizeHolder, **1**);

bufferSizeHolder += **1**;

printf("Site Name Size = %d**\n**",decoded\_site\_name\_size );

decoded\_site\_name = (**char**\*)malloc(**sizeof**(**char**)\*decoded\_site\_name\_size);

memcpy(decoded\_site\_name, receivedBuffer + bufferSizeHolder, decoded\_site\_name\_size);

bufferSizeHolder += decoded\_site\_name\_size;

printf("Site Name = %s**\n**",decoded\_site\_name );

memcpy( & decoded\_site\_desc\_size, receivedBuffer + bufferSizeHolder, **1**);

bufferSizeHolder += **1**;

printf("Site Description Size = %d**\n**",decoded\_site\_desc\_size );

decoded\_site\_desc = (**char**\*)malloc(decoded\_site\_desc\_size);

memcpy(decoded\_site\_desc, receivedBuffer + bufferSizeHolder, decoded\_site\_desc\_size);

bufferSizeHolder += decoded\_site\_desc\_size;

printf("Site Decription = %s**\n**", decoded\_site\_desc);

memcpy( & decoded\_station\_count, receivedBuffer + bufferSizeHolder, **1**);

bufferSizeHolder += **1**;

printf("Station Count = %d**\n**", decoded\_station\_count);

decoded\_station\_name\_size =(**uint8\_t**\*) malloc(**sizeof**(**uint8\_t**)\*decoded\_station\_count);

decoded\_station\_name = (**char** \*\*) malloc(decoded\_station\_count \* **sizeof**(**char**\*));

decoded\_multicast\_address = (**uint32\_t** \*) malloc(decoded\_station\_count \* **sizeof**(**uint32\_t**));

decoded\_data\_port = (**uint16\_t** \*) malloc(decoded\_station\_count \* **sizeof**(**uint16\_t**));

decoded\_info\_port = (**uint16\_t** \*) malloc(decoded\_station\_count \* **sizeof**(**uint16\_t**));

decoded\_bit\_rate = (**uint32\_t** \*) malloc(decoded\_station\_count \* **sizeof**(**uint32\_t**));

**for** (tempx = **0**; tempx < decoded\_station\_count; tempx++) {

printf("Station number = %d**\n**",tempx+**1**);fflush(stdout);

bufferSizeHolder += **1**;

printf("bufferSizeHolder = %d**\n**",bufferSizeHolder);fflush(stdout);

memcpy( & decoded\_station\_name\_size[tempx], receivedBuffer + bufferSizeHolder, **1**);

bufferSizeHolder += **1**;

printf("Station name size = %d**\n**",decoded\_station\_name\_size[tempx]);fflush(stdout);

// printf("Station name size = %d\n",\*(receivedBuffer + bufferSizeHolder-1));fflush(stdout);

// decoded\_station\_name[tempx] = (char\*)malloc(sizeof(char)\*decoded\_station\_name\_size);

decoded\_station\_name[tempx] = (**char**\*)malloc(**sizeof**(**char**)\*decoded\_station\_name\_size[tempx]);

memcpy(decoded\_station\_name[tempx], receivedBuffer + bufferSizeHolder, decoded\_station\_name\_size[tempx]);

bufferSizeHolder += decoded\_station\_name\_size[tempx];

printf("Station name = %s**\n**",decoded\_station\_name[tempx] );fflush(stdout);

// printf("Station name = %c\n", \*(receivedBuffer + bufferSizeHolder - decoded\_station\_name\_size[tempx]));

memcpy( & decoded\_multicast\_address[tempx], receivedBuffer + bufferSizeHolder, **4**);

bufferSizeHolder += **4**;

decoded\_multicast\_address[tempx] = ntohs(decoded\_multicast\_address[tempx]);

printf("Multicast address = %d**\n**",decoded\_multicast\_address[tempx] );

memcpy( & decoded\_data\_port[tempx], receivedBuffer + bufferSizeHolder, **2**);

bufferSizeHolder += **2**;

decoded\_data\_port[tempx] = ntohs(decoded\_data\_port[tempx]);

printf("Data port = %d**\n**",decoded\_data\_port[tempx] );

memcpy( & decoded\_info\_port[tempx], receivedBuffer + bufferSizeHolder, **2**);

bufferSizeHolder += **2**;

decoded\_info\_port[tempx] = ntohs(decoded\_info\_port[tempx]);

printf("Info port = %d**\n**",decoded\_info\_port[tempx] );

memcpy( & decoded\_bit\_rate[tempx], receivedBuffer + bufferSizeHolder, **4**);

bufferSizeHolder += **4**;

decoded\_bit\_rate[tempx] = ntohs(decoded\_bit\_rate[tempx]);

printf("Bit rate = %d**\n**",decoded\_bit\_rate[tempx] );

}

// printf("decoded\_station\_count = %d\n",decoded\_station\_count);

/\*2) Display recv\_buf->station\_list[] and get user input and set up a multicast recv socket(s)\*/

puts("");

puts("The stations are:");

**for** (tempx = **0**; tempx < decoded\_station\_count; tempx++) {

printf("**\n**%d.**\t**%s", tempx, decoded\_station\_name[tempx]);

}

puts("");

puts("");

puts("");

fflush(stdout);

// if(scanfCounter == -1){

// puts("Enter preferred station :");

// fflush(stdout);

// scanf(" %d",&preferred\_station\_number);

// scanfCounter = 1;

// }

//scanf(" %d",&preferred\_station\_number);

//char ch[2];

//printf("Press Y to continue:\n");

/\*do{

fflush(stdout);

// printf("Press Y to continue:");fflush(stdout);fflush(stdin);

// scanf(" %c",&ch);

ch[0] = fgetc(stdin);

fputc(ch[0],stdout);

fflush(stdout);

ch[1] = fgetc(stdin);

fputc(ch[1],stdout);

fflush(stdout);

if(ch[0]=='Y'||ch[1]=='Y')

break;

// ch = fgetc(stdin);

// printf("This is the fucking character = %d",ch);fflush(stdout);

fflush(stdin);

}while(ch[0] !='Y' || ch[1] != 'Y');

ch[0] = 'N';

\*/// while((ch = fgetc(stdin)) != 'Y');

puts("Enter preferred station:");

scanf(" %d",&preferred\_station\_number);

printf("Preferred station number is %d**\n**",preferred\_station\_number+**10**);

sprintf(default\_mcast\_addr,"239.192.29.%d",preferred\_station\_number+**10**);

puts(default\_mcast\_addr);

**FORK:**forkResult = fork();

**if**(DEBUG2 == **1**)

printf("after forking = %d",forkResult);

**if**(forkResult == **0**){

**if**(DEBUG2 == **1**)

puts("inside fork child");

**if**(DEBUG2 == **1**)

printf("pid = %d**\n**",getpid());

/\* create socket \*/

// puts("sending Multicast join request");

**if** ((s = socket(PF\_INET, SOCK\_DGRAM, **0**)) < **0**) {

perror("receiver: socket");

exit(**1**);

}

/\* build address data structure \*/

memset((**char** \* ) & sin, **0**, **sizeof**(sin));

sin.sin\_family = AF\_INET;

sin.sin\_addr.s\_addr = htonl(INADDR\_ANY);

sin.sin\_port = htons(MC\_PORT);

/\*Use the interface specified \*/

memset( & ifr, **0**, **sizeof**(ifr));

strncpy(ifr.ifr\_name, if\_name, **sizeof**(if\_name) - **1**);

/\*if ((setsockopt(s, SOL\_SOCKET, SO\_BINDTODEVICE, (void \* ) & ifr,

sizeof(ifr))) < 0) {

perror("receiver: setsockopt() error");

close(s);

exit(1);

}\*/

/\* bind the socket (MULTICAST)\*/

**if** ((bind(s, (**struct** sockaddr \* ) & sin, **sizeof**(sin))) < **0**) {

perror("receiver: bind()");

exit(**1**);

}

**if**(DEBUG2 == **1**)

puts("after bind to multicast");

/\* Multicast specific code follows \*/

/\* build IGMP join message structure \*/

mcast\_req.imr\_multiaddr.s\_addr = inet\_addr(default\_mcast\_addr); //default\_mcast\_addr

mcast\_req.imr\_interface.s\_addr = htonl(INADDR\_ANY);

/\* send multicast join message \*/

**if** ((setsockopt(s, IPPROTO\_IP, IP\_ADD\_MEMBERSHIP,

(**void** \* ) & mcast\_req, **sizeof**(mcast\_req))) < **0**) {

perror("mcast join receive: setsockopt()");

exit(**1**);

}

**if**(DEBUG2 == **1**)

puts("before file handling");

// char\* temp\_filename = malloc(sizeof(char)\*10);

// sprintf(temp\_filename,"%d.mp3",file\_counter);

**FILE** \*fp\_recv = fopen("tempHolder.mp3","wb");

**if**(fp\_recv == NULL){

**if**(DEBUG2 == **1**)

puts("file pointer is null");

}

**int** once=**0**,l=**0**;

/\* receive multicast messages \*/

**while**(**1**){

/\* reset sender struct \*/

memset(&mcast\_saddr, **0**, **sizeof**(mcast\_saddr));

mcast\_saddr\_len = **sizeof**(mcast\_saddr);

/\* clear buffer and receive \*/

memset(buffer\_recv, **0**, **sizeof**(buffer\_recv));

// add file handling instead of printing the buf and also unpack the correct struct

**if**(DEBUG2 == **1**)

puts("waiting on recv");

**if** ((len = recvfrom(s, buffer\_recv, BUF\_SIZE, **0**, (**struct** sockaddr\*)&mcast\_saddr,

&mcast\_saddr\_len)) < **0**) {

puts(buffer\_recv);

perror("receiver: recvfrom()");

exit(**1**);

}

**if**(DEBUG2 == **1**)

printf("pid = %d**\n**",getpid());

**if**(DEBUG2 == **1**)

printf("Length : %d",len);

// puts(buffer\_recv);

// puts("inside while after recv");

// fputs(buffer\_recv, fp\_recv);

once++;

// printf("once = %d\n",once);

fflush(stdout);

**for**(l=**0**;l<len;l++){

//puts("inside writing");

//fputc(buffer\_recv[l],stdout);

fflush(fp\_recv);

fputc(buffer\_recv[l],fp\_recv);

fflush(fp\_recv);

}

**if**(once == **15**){

**if**(DEBUG2 == **1**)

puts("once==25");

system("ffmpeg -loglevel fatal -i tempHolder.mp3 -f mp2 - | ffplay - &");

}

fflush(stdout);

fflush(fp\_recv);

}

}

**else**

{

**char** ch;

// printf("\n pid before switch = %d\n ",forkResult);

**while**(ch!='X'){

printf(" Press P to Pause **\n**");

printf(" Press R to Reset **\n**");

printf(" Press C to Change Station **\n**");

printf(" Press X to exit **\n**");

fflush(stdin);

//scanf("%c",&ch);

scanf(" %c",&ch);

/\*do{

ch = fgetc(stdin);

}while(ch=='\n');\*/

// while((ch = fgetc(stdin)) == '\n');

fflush(stdin);

ch = toupper(ch);

printf("**\n**The input character is : %c**\n**",ch);

fflush(stdout);

**switch**(ch){

**case** 'P':

kill(forkResult, SIGTERM);

system("killall ffplay");

system("rm tempHolder.mp3");

// system("clear");

//prepare penetration

**break**; //sigkill child

**case** 'R':

// kill(forkResult, SIGCONT);

// system("clear");

**goto** FORK;

//set variable to penetrate

**break**; //fork

**case** 'C':

system("killall ffplay");

system("rm tempHolder.mp3");

kill(forkResult, SIGTERM);

// system("clear");

// puts("Enter preferred station :\n");

// scanf(" %d",&preferred\_station\_number);

**goto** CHANGE;

**break**;

**case** 'X':

system("killall ffplay");

system("killall a.out");

// system("clear");

exit(**0**);

**break**;

**default** :

// puts("in default");

**break**;

}

fflush(stdout);

fflush(stdin);

}

}

close(s);

}

**return** **0**;

}

### Receiver Side:

/\* CSD 304 Computer Networks, Fall 2016

Sender

Team: Prasanna Natarajan

Siddhart Mitra

Sridhar Renga Ramanujam

Vedant Chakravarthy

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <time.h>

#define MC\_PORT 7779

#define BUF\_SIZE 64480

#define TCP\_PORT 5432

#define MAX\_PENDING 10

#define MAX\_NUM\_OF\_STATIONS 2

#define MAX\_FILES\_IN\_A\_STATION 1

#define DEBUG 0

/\*Global Variables\*/

**int** tempSizeHolder = **0**;

**typedef** **struct** station\_info {

**uint8\_t** station\_number;

**uint8\_t** station\_name\_size;

**char** \* station\_name;

**uint32\_t** multicast\_address;

**uint16\_t** data\_port;

**uint16\_t** info\_port;

**uint32\_t** bit\_rate;

}

**station\_info\_t**;

**void** **serialize\_station\_info\_t**(**station\_info\_t** \*buf,**char** \*x){

**char** buffer[**2048**]; //ARBITRARY SIZE INITIALIZATION

**if**(DEBUG == **1**)

puts("Inside serialize\_station\_info\_t");

memcpy(buffer ,&(buf->station\_number) ,**1**); //1

memcpy(buffer+**1** ,&(buf->station\_name\_size) ,**1**); //2

memcpy(buffer+**2** ,(buf->station\_name) ,buf->station\_name\_size); //3

**uint32\_t** multicast\_address = htons(buf->multicast\_address); //4

memcpy(buffer+**2**+buf->station\_name\_size ,&multicast\_address ,**4**); //5

**if**(DEBUG == **1**)

puts("5");

**uint16\_t** data\_port = htons(buf->data\_port); //6

memcpy(buffer+**2**+buf->station\_name\_size+**4** ,&data\_port ,**2**); //7

**if**(DEBUG == **1**)

puts("6");

**uint16\_t** info\_port = htons(buf->info\_port); //8

memcpy(buffer+**2**+buf->station\_name\_size+**4**+**2** ,&info\_port ,**2**);//9

**if**(DEBUG == **1**)

puts("9");

**uint32\_t** bit\_rate = htons(buf->bit\_rate); //10

memcpy(buffer+**2**+buf->station\_name\_size+**4**+**2**+**2** ,&bit\_rate ,**4**);//11

**if**(DEBUG == **1**)

puts("11");

tempSizeHolder = **2**+buf->station\_name\_size+**4**+**2**+**2**+**4**; //12

**if**(DEBUG == **1**)

puts("12");

memcpy(x,buffer,tempSizeHolder);

**if**(DEBUG == **1**)

puts("End of serialize\_station\_info\_t");

}

**typedef** **struct** site\_info {

**uint8\_t** type;

**uint8\_t** site\_name\_size;

**char** \* site\_name;

**uint8\_t** site\_desc\_size;

**char** \* site\_desc;

**uint8\_t** station\_count;

**station\_info\_t** \* station\_list;

}

**site\_info\_t**;

**void** **serialize\_site\_info\_t**(**site\_info\_t** \*buf,**char** \*x){

**char** buffer[**1024**]; //ARBITRARY SIZE INITIALIZATION

memcpy(buffer ,&(buf->type) ,**1**);

memcpy(buffer+**1** ,&(buf->site\_name\_size) ,**1**);

memcpy(buffer+**2** ,(buf->site\_name) ,buf->site\_name\_size);

memcpy(buffer+**2**+buf->site\_name\_size ,&(buf->site\_desc\_size) ,**1**);

memcpy(buffer+**2**+buf->site\_name\_size+**1** ,(buf->site\_desc) ,buf->site\_desc\_size);

memcpy(buffer+**2**+buf->site\_name\_size+**1**+buf->site\_desc\_size,&(buf->station\_count),**1**);

**char**\* otherStruct;

otherStruct = (**char**\*)malloc(**20000**);

**if**(DEBUG == **1**)

puts("In the middle of serialize\_site\_info\_t");

// FATAL ERROR

**int** sumSizes = **0**;

**int** tempx;

**for**(tempx = **0**; tempx < MAX\_NUM\_OF\_STATIONS; tempx++){ // station count has been replaced by MAX\_NUM\_Of\_STATIONS

**if**(DEBUG == **1**)

puts("Before serialize\_site\_info\_t inside serialize\_site\_info\_t");

**serialize\_station\_info\_t**(&(buf->station\_list[tempx]),otherStruct);

**if**(DEBUG == **1**)

puts("After serialize\_site\_info\_t inside serialize\_site\_info\_t");

sumSizes+=tempSizeHolder;

memcpy(buffer+**2**+buf->site\_name\_size+**1**+buf->site\_desc\_size+**1**+sumSizes,otherStruct,tempSizeHolder);

}

**if**(DEBUG == **1**)

puts("End of serialize\_site\_info\_t");

memcpy(x,buffer,**sizeof**(buffer));

}

**typedef** **struct** station\_not\_found {

**uint8\_t** type;

**uint8\_t** station\_number;

}

**station\_not\_found\_t**;

**void** **serialize\_station\_not\_found\_t**(**station\_not\_found\_t** \*buf,**char** \*x){

**char** buffer[**5**]; //ARBITRARY SIZE INITIALIZATION

memcpy(buffer ,&(buf->type) ,**1**);

memcpy(buffer+**1** ,&(buf->station\_number) ,**1**);

memcpy(x,buffer,**sizeof**(buffer));

}

**typedef** **struct** song\_info {

**uint8\_t** type;

**uint8\_t** song\_name\_size;

**char**\* song\_name;

**uint16\_t** remaining\_time\_in\_sec;

**uint8\_t** next\_song\_name\_size;

**char**\* next\_song\_name;

}

**song\_info\_t**;

**void** **serialize\_song\_info\_t**(**song\_info\_t** \*buf,**char** \*x){

**char** buffer[**1024**]; //ARBITRARY SIZE INITIALIZATION

memcpy(buffer ,&(buf->type) ,**1**);

memcpy(buffer+**1** ,&(buf->song\_name\_size) ,**1**);

memcpy(buffer+**2** ,(buf->song\_name) ,buf->song\_name\_size);

**uint16\_t** remaining\_time\_in\_sec = htons(buf->remaining\_time\_in\_sec);

memcpy(buffer+**2**+buf->song\_name\_size ,&remaining\_time\_in\_sec ,**2**);

memcpy(buffer+**2**+buf->song\_name\_size+**2** ,&(buf->next\_song\_name\_size),**1**);

memcpy(buffer+**2**+buf->song\_name\_size+**2**+**1** ,(buf->next\_song\_name) ,buf->next\_song\_name\_size);

memcpy(x,buffer,**sizeof**(buffer));

}

/\*Global Variables\*/

**char**\* all\_mcast\_addresses[MAX\_FILES\_IN\_A\_STATION];

**int** **main**(**int** argc, **char** \* argv[]) {

**int** s, s\_mcast ,new\_s; /\* socket descriptor \*/

**struct** sockaddr\_in sin; /\* socket struct for TCP\*/

**struct** sockaddr\_in sin\_mcast; /\*socket struct for multi cast\*/

**char** buf[BUF\_SIZE];

**int** len;

**pid\_t** pid;

/\* Multicast specific \*/

**char** \* mcast\_addr; /\* multicast address \*/

**char** \* team\_multicast\_address = "239.192.29.10";

**struct** timespec tim,tim2;

tim.tv\_sec = **1**;

tim.tv\_nsec = **0**;

**int** length = **0**;

**int** looper = **0**;

/\* Add code to take port number from user \*/

**if** (argc == **2**) {

mcast\_addr = argv[**1**];

} **else** {

fprintf(stderr, "usage: sender multicast\_address**\n**");

exit(**1**);

}

**struct** hostent \*hp;

hp = gethostbyname(mcast\_addr);

**if**(!hp){

perror("simplex-talk: Unknown Host");

exit(**1**);

}

**else**{

printf("Server's remote host : %s**\n**",mcast\_addr);

}

/\*Filling up the mcast\_addresses\*/

**int** j;

**for** (j = **0**; j < MAX\_NUM\_OF\_STATIONS; ++j)

{

all\_mcast\_addresses[j] = malloc(**sizeof**(**char**)\***15**);

**char** ip[**3**];

sprintf(ip,"%d",**10**+j);

strcat(all\_mcast\_addresses[j],"239.192.29.");

strcat(all\_mcast\_addresses[j],ip);

puts(all\_mcast\_addresses[j]);

}

pid = fork();

**if** (pid == **0**) {

/\* Send multicast messages \*/

/\* Warning: This implementation sends strings ONLY \*/

/\* You need to change it for sending A/V files \*/

memset(buf, **0**, **sizeof**(buf));

**char** str[INET\_ADDRSTRLEN];

/\* setup passive open \*/

/\* build address data structure \*/

memset((**char** \* ) & sin, **0**, **sizeof**(sin));

sin.sin\_family = AF\_INET;

sin.sin\_addr.s\_addr = inet\_addr(mcast\_addr); //This is correct, naming is misleading

sin.sin\_port = htons(TCP\_PORT);

**if** ((s = socket(PF\_INET, SOCK\_STREAM, **0**)) < **0**) {

perror("simplex-talk: socket");

exit(**1**);

}

inet\_ntop(AF\_INET, & (sin.sin\_addr), str, INET\_ADDRSTRLEN);

printf("Server is using address %s and port %d.**\n**", str, TCP\_PORT);

/\*

int yes=1;

//char yes='1'; // Solaris people use this

// lose the pesky "Address already in use" error message

if (setsockopt(s,SOL\_SOCKET,SO\_REUSEADDR,&yes,sizeof yes) == -1) {

perror("setsockopt");

exit(1);

} \*/

**int** yes=**1**;

//char yes='1'; // Solaris people use this

// lose the pesky "Address already in use" error message

**if** (setsockopt(s,SOL\_SOCKET,SO\_REUSEADDR,&yes,**sizeof** (yes)) == -**1**) {

perror("setsockopt");

exit(**1**);

}

**if** ((bind(s, (**struct** sockaddr \* ) &sin, **sizeof**(sin))) < **0**) {

perror("simplex-talk: bind");

exit(**1**);

} **else**

printf("Server bind done.**\n**");

listen(s, MAX\_PENDING);

**if** ((new\_s = accept(s, (**struct** sockaddr\*) &sin, &len)) < **0**) {

perror("simplex-talk: accept");

exit(**1**);

}

**else**{

puts("inside else");

}

puts("after listen");

**while** (**1**) {

**if**(DEBUG == **1**)

puts("inside while");

/\*

check station list and then send current station list and break

3) fill the struct

4) serialize the struct

5) send it to the reciever using syntax send(new\_s,buff,strlen(buff)+1,0);

\*/

**if**(DEBUG == **1**)

puts("Begin things");

**char**\* station\_list\_array[MAX\_NUM\_OF\_STATIONS];

**int** j;

**for**(j=**0**;j<MAX\_NUM\_OF\_STATIONS;j++){

station\_list\_array[j] = malloc(**sizeof**(**char**)\***20**);

strcat(station\_list\_array[j],"channel");

**char** j\_char[**33**];

sprintf(j\_char,"%d",j);

strcat(station\_list\_array[j],j\_char);

}

**if**(DEBUG == **1**)

puts("Intermediate places");

**site\_info\_t** \*first\_send = malloc(**sizeof**(first\_send)\***sizeof**(**char**));

first\_send->type = **10**;

first\_send->site\_name\_size= **5**;

first\_send->site\_name = malloc(**sizeof**(**char**)\*first\_send->site\_name\_size);

strcpy(first\_send->site\_name, "Mitra");

first\_send->site\_desc\_size = **20**;

first\_send->site\_desc = malloc(**sizeof**(**char**)\*first\_send->site\_desc\_size);

strcpy(first\_send->site\_desc,"Mitra's funny videos");

first\_send->station\_count = MAX\_NUM\_OF\_STATIONS;

**station\_info\_t** stations[MAX\_NUM\_OF\_STATIONS];

// puts("Populate station list");

**for**(j=**0**;j<MAX\_NUM\_OF\_STATIONS;j++){

// stations[j] = (station\_info\_t)malloc(sizeof(station\_info\_t)\*sizeof(char));

stations[j].station\_number = j;

stations[j].station\_name\_size = strlen(station\_list\_array[j]);

stations[j].station\_name = malloc(**sizeof**(**char**)\*strlen(station\_list\_array[j]));

stations[j].station\_name = station\_list\_array[j];

stations[j].multicast\_address = j;

stations[j].data\_port = **5555**+j;

stations[j].info\_port = **6666**+j;

stations[j].bit\_rate = **126**;

}

**if**(DEBUG == **1**)

puts("Final spaces");

fflush(stdout);

first\_send->station\_list = stations;

**if**(DEBUG == **1**)

puts("Before malloc");

**char**\* x = malloc(**sizeof**(**char**)\***20000**);

**if**(DEBUG == **1**)

puts("after malloc");

**serialize\_site\_info\_t**(first\_send,x);

**if**(DEBUG == **1**){

puts("The serialized output is");

puts(x);

puts("type : ");

printf("%d",x[**0**]);

puts("site\_name\_size : ");

printf("%d",x[**1**]);

}

**if**(DEBUG == **1**)

puts("Sending thing");

send(new\_s,x,strlen(x)+**1**,**0**);

**if**(DEBUG == **1**)

puts("Successful send");

}

} **else** {

// while(1)

**if**(DEBUG == **1**)

puts("inside first else");

**int** station\_number;

**pid\_t** pid1;

**int** xx;

**for**(station\_number = **0**;station\_number<MAX\_NUM\_OF\_STATIONS;station\_number++){

pid1 = fork();

**if**(pid1==**0**){

**if**(DEBUG == **1**)

puts("inside fork");

**if**(DEBUG == **1**)

printf("fork number = %d",station\_number);

fflush(stdout);

**if** ((s\_mcast = socket(PF\_INET, SOCK\_DGRAM, **0**)) < **0**) {

perror("server: socket");

exit(**1**);

}

/\* build address data structure \*/

memset((**char** \* ) & sin, **0**, **sizeof**(sin\_mcast));

sin\_mcast.sin\_family = AF\_INET;

sin\_mcast.sin\_addr.s\_addr = inet\_addr(all\_mcast\_addresses[station\_number]); //mcast\_addr

sin\_mcast.sin\_port = htons(MC\_PORT);

printf("the all\_mcast\_addresses[station\_number] = %s",all\_mcast\_addresses[station\_number]);

fflush(stdout);

**char**\* filenames\_in\_a\_station[MAX\_FILES\_IN\_A\_STATION]; // each string will contain all the names of that particular staion

**int** i;

**char** command\_ls[**20**] = "ls channel";

**char** i\_char[**33**];

sprintf(i\_char,"%d/",station\_number);

strcat(command\_ls,i\_char);

//printf("%s",command\_ls);

strcat(command\_ls," | grep .mp3");

//printf("%s\n",command\_ls);

**FILE** \* temp\_fp = popen(command\_ls,"r");

**for** (i = **0**; i < MAX\_FILES\_IN\_A\_STATION; ++i)

{

filenames\_in\_a\_station[i] = malloc(**sizeof**(**char**)\***100**);

}

**char** \* temp\_filename = malloc(**sizeof**(**char**)\***20**);

**char** \* channel\_name = malloc(**sizeof**(**char**)\***20**);

sprintf(channel\_name,"./channel%d/",station\_number);

i=**0**;

**while**(fgets(temp\_filename,**20**,temp\_fp)!=NULL){

**if**(temp\_filename[strlen(temp\_filename)-**1**]=='\n'){

temp\_filename[strlen(temp\_filename)-**1**]='\0';

**if**(DEBUG == **1**)

puts("inside if");

}

strcat(channel\_name,temp\_filename);

strcat(filenames\_in\_a\_station[i],channel\_name);

puts(filenames\_in\_a\_station[i]);

i++;

}

**char** buffer\_to\_send\_multcast[BUF\_SIZE];

**FILE** \*fp\_array[MAX\_FILES\_IN\_A\_STATION];

**FILE** \*fp\_write = fopen("temp.mp3","w");

puts("just before sending multicast");

**for**(i=**0**;i< MAX\_FILES\_IN\_A\_STATION;i++){

printf("inside for in sending multicast = %d",i);

fflush(stdout);

fp\_array[i] = fopen(filenames\_in\_a\_station[i],"r");

puts("after creating file pointer");

**if**(fp\_array[i] == NULL){

printf("**\n**%s**\n**",filenames\_in\_a\_station[i]);

fflush(stdout);

puts("no file");

**break**;

}

fflush(stdout);

fflush(stdin);

fseek(fp\_array[i], **0**, SEEK\_END);

length = ftell(fp\_array[i]);

fseek(fp\_array[i], **0**, **0**);

fflush(fp\_array[i]);

**for**(looper = **0**;looper<length;looper++){

buffer\_to\_send\_multcast[looper%BUF\_SIZE] = fgetc(fp\_array[i]);

**if**((looper%BUF\_SIZE == BUF\_SIZE-**1**)){

fwrite(buffer\_to\_send\_multcast,BUF\_SIZE,**1**,fp\_write);

**if** ((len = sendto(s\_mcast, buffer\_to\_send\_multcast, BUF\_SIZE, **0**,(**struct** sockaddr \*)&sin\_mcast, **sizeof**(sin\_mcast))) == -**1**) {

perror("sender: sendto");

exit(**1**);

}

puts("Printed");

nanosleep(&tim,&tim2);

}

}

i=(i+**1**)%MAX\_FILES\_IN\_A\_STATION;

}

}**else**{

// int status;

// pid1 = wait(&status);

}

}

pid = wait();

}

close(s);

**return** **0**;

}