Centralized Speech Enhancement for Audio Conferencing

Ali Ashique
Department of Electronics
Quaid-i-Azam University, Islamabad
Pakistan

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Science
in
Electronics
July, 2019

DEPARTMENT OF ELECTRONICS
QUAID-I-AZAM UNIVERSITY
ISLAMABAD

A thesis entitled "Centralized Speech Enhancement for Audio Conferencing" by Ali Ashique in partial fulfillment of the requirements for the degree of Bachelor of Science, has been approved and accepted by the following

Dr. Muhammad Zia
Associate Professor
Department of Electronics
Quaid-i-Azam University, Islamabad Pakistan

Dr. Syed Aqeel Abbas Bukhari
chairman
Department of Electronics
Quaid-i-Azam University, Islamabad Pakistan

Dedicated to my parents for their unconditional love, care and continuous encouragement throughout my life and to my supervisor Dr. Muhammad Zia for his continuous encouragement, support and intuitive guide.

Acknowledgements

All praises to Allah for the strengths and His blessings throughout my life.

All your dream needs is someone to believe in it and that is the credit of my supervisor Dr. Muhammad Zia throughout my dissertation. Without him, I even cannot imagine to think in this domain. I express my deepest gratitude to him for his constant support, guidance, motivation and help during my BS. Discussions with him in every field of life are always fruitful to a remarkable extent. I will always remember his calm and tolerant and down to earth nature.

I am greatly indebted to all the faculty members of the Department of Electronics, QAU especially Dr. M. Naeem Ali Bhatti, Dr. M. Aqueel Ashraf, Dr. Aqeel Abbas Bukhari, Dr. Qaisar Abass Naqvi, Dr. Azhar Abass Rizvi, Dr. Farhan Saif and Dr. Qurat-ul-Ain Minhas for boosting my morale throughout the studies. They have always been caring, a source of wisdom and motivation.

A special thanks to my senior Awais Ahmed and Khubaib Ali Kiyani for their love, encouragement and guidance throughout my university life. Hats off regards to Saadia Zainab, Rabia Shamim, Farooq Ahmed best lab mate Nawal Naeem, Noor ul Amin and especially to the most gracious seniors Hassan Ullah and Zohaib for their continuous support and everlasting company in every ups and downs.

Last and most special thanks to my parents for their love and countless efforts to make me reach this remarkable position in life. My heartiest gratitude goes to my siblings for their love, encouragement, and support they have provided throughout my life and most special thanks to Rabeea Basir who introduced me to signal processing and especially to internet of things (IoT).

Ali Ashique

July , 2019

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

DMA	Direct Memory Access
A/D	Analog to Digital converter
D/A	Digital to Analog converter
S8	signed 8 bit
U8	unsigned 8 bits
API	Application Interface
PCM	Pulse Code Modulation
IP	Internet Protocol
RTP	Real time Transport Protocol
UDP	User Data gram Protocol
TCP	Transport Control Protocol

ABSTRACT

Analysis of a Speech signal is one of the important areas of research in multimedia application. One of the main interests behind analyzing the speech signal is quality. Quality of a Speech signal and its intelligibility degrade when noise mixed with speech and it becomes more difficult to extract the desired information contained in the signal. When we talk about enhancement in the Speech signal, Reduction or removal of unwanted noise is very important from the speech signal in order to get the desired information contained in Speech signal. This project proposes an algorithm for removing the noise effect from Speech signal. In order to validate our approach for noise suppression, we have first implemented it on MATLAB then on C language for Real-time operating system (RTOS).We investigated the noise suppression algorithm on Human voice and compared the results with noisy Speech signal. The proposed result shows that this approach makes efficient use of VoIP with speech enhancement for Audio conferencing. Voice over internet protocol (VoIP) relates with the transmission of a data like voice over network. This application is cost effective for communication with the help of the existing network. VoIP is a modern technology by which user can easily communicate with any one regardless of distance. VoIP application can run through any local or shared network. Traditional telephone uses public switched telephone network. But VoIP application use the channels over internet by getting the digitized input data, and form those data packets, then transmit these packets over internet channels. VoIP application is trending now a days because it is cost saving application and is the reason behind its popularity. VoIP gives many options for the communication on existing network which traditional phones do not. In this project, we develop VoIP proprietary protocol for Ubuntu operating system.

CHAPTER 1

INTRODUCTION

1.1 SPEECH ENHANCEMENT USING NOISE SUPPRESSION:

To improve the sound quality by eliminating the noise from desired signal. Noise estimator is a basic part to improve sound quality. The Noise Spectrum Estimator estimated the noise from noisy speech signal during non-speech activity. While making Noise Spectrum Estimator some of the assumptions were used in developing analysis. The environmental noise digitally or acoustically added to the Speech signal. The environmental noise remains stationary during speech activity. If the background noise changes to the new stationary state, there will be enough time (approximately 400ms) for Noise Estimator to estimate the new level of noise during non-speech activity. For detection of Speech, Voice active Detector (VAD) detects the existence of Speech in a signal. The other tool to detect the noise from signal is Zero Crossing Detector. To detect the noise we know noise has higher rate of Zero-Crossing, so we can easily subtract the Noise from the Speech signal and easily achieve the Enhanced Speech for Audio Conferencing.

Why we use VoIP?

- To validate the Speech enhancement algorithm for Audio Conferencing
- It provides the cheaper telecommunication.
- No need of wired connections.
- Cost effective.
- Easy to use for Audio conferencing.

1.2 OBJECTIVES:

We use real time audio algorithm to design Speech Enhancement for audio conferencing VOIP Application. Noise spectrum creates the annoying effect which is called background noise. The Noise suppression technique seems to allow us to remove the fair amount of Noise from desired signal.

Spectral suppression technique seems to let us achieve the better acoustic Noise reduction. To achieve the spectral subtraction we have to estimate the Noise. To achieve the good estimate of Noise we take a non-speech part of a signal which has only noise to design a filter. The approach we use to detect the non-speech activity is Voice Active Detector (VAD), which can help us to find the speech activity without explicitly identifying the non-speech part. Our goal is to design a simple algorithm which discretizes the amplitude range of a noise, so we can easily detect the most frequently occurring amplitude level to consider it as estimate of Noise. We report our project as Centralized Speech Enhancement for Audio Conferencing. Our project targets the problem of Speech Enhancement and all our experiment is on Speech Signal.

1.3 IMPLEMENTATION:

For implementation we use gedit which is editor in Linux. The API we use for audio application is ALSA. The following major step shows how we achieve Enhanced Speech.

1. Take the Speech signal.
2. Speech signal must have some noise, which need to be removed at the receiver end for audio Conferencing.
3. Speech and Noise can be detected by using VAD and Zero Crossing tool.
4. In frequency domain it is easy to analyze the signal, for the removal of noise we first convert it into frequency domain by using FFT.
5. Using Noise estimator we can estimate the unwanted part (Noise) in a signal.
6. For removing of Noise we used frame by frame analysis on basis of SNR.
7. High SNR (Signal to noise ratio). If ratio is higher than 1 means, more signal than noise.
8. Low SNR. If ratio is less than 1 means, more noise than Speech.
9. Zero SNR means signal is pure
10. Filtering and Smoothing.
11. Enhanced audio

CHAPTER 2

INTRODUCTION TO ALSA

In this project, we actually work in audio through VoIP. There are many sound architecture use for sound programming, but we use ALSA. ALSA abbreviated as Advanced Linux Sound Architecture. For sound programming, we use ALSA API and other utility program for designing under Linux environment. For ALSA based project, we focus on the PCM interfaces of ALSA. Basically ALSA API is not only the one sound API which is available in Linux. But for the purpose of low level programming ALSA is the best choice for maximum control under user.

2.1 BACKGROUND

ALSA project was begun to control the sound drivers in Linux kernel, because it were lagging behind the abilities of new sound application. ALSA was merged into the Linux kernel source after release of 2.6 kernel.

2.2 BASICS OF DIGITAL AUDIO

The digital signal can be obtained from ADC converter for the further process of audio control. Sound is varying behavior because of air pressure which is in the form of waves which is converted to electrical form with the help of transducer likewise microphone. ADC converter helps to convert the analog signal to digital signal. From digital signal we obtain discrete values and each discrete value called a sample, at a regular interval of time known as sampling rate. For the output by sending samples towards DAC converter and output transducer like loud speaker, the voice can be reproduced. The samples size can be expressed in bits which determine how perfectly the voice is in digital form. Sampling rate plays important role for the sound quality.

2.3 BASICS REGARDING ALSA

ALSA has many kernel device drivers for different sound cards also it provides ALSA API library called libasound. Application related to the voice can be programmed using API library. And this library gives friendly programming interface containing logical names of devices so that the user do not worry to learn low level programming in details. ALSA gives a number of command-line utilities, including a mixer, sound file player and tools for controlling special features of specific sound cards.

2.4 Architecture of ALSA

ALSA API consists of many interfaces it supports which are given below:

- Control interface
- PCM interface
- Raw MIDI interface
- Timer interfaces
- Sequencer interface
- Mixer interface

2.5 ALSA DEVICE NAMES

The ALSA API works with the logical device name instead of device file such as hardware devices or plugins. The format use for the hardware device is hw:i,j.

- 'i' represents the card number .
- 'j' represents the device in which card installed.

hw: 0,0 is the first sound device. Plugins like plughw: which gives access to the hardware device but provide features for example conversion of sampling rate.

2.6 ALSA SOUND BUFFERS

Sound card consists of hardware buffer which stores samples which are recorded. When buffer begins to full then interrupt call generate. DMA used by sound driver to transfer samples to application in memory likewise for playback another application buffer is transferred from memory to the sound card’s hardware buffer using DMA. Hardware buffer called ring buffers, ring buffers means the data wraps back to the start when the buffer is ended. A pointer is used to keeps track both hardware and application buffer’s current position. The buffer size can be easily adjusted by libasound calls which are ALSA library. If the buffer size is large it causes unacceptable delays in results while transferring one operation. So ALSA provides solution to breakdown the buffer into series of periods which can be called fragments. Periods have frames and a frame contains the samples values at one point. If we use stereo device frame contains two channels.

For further understanding Figure 1 illustrates the buffer understanding

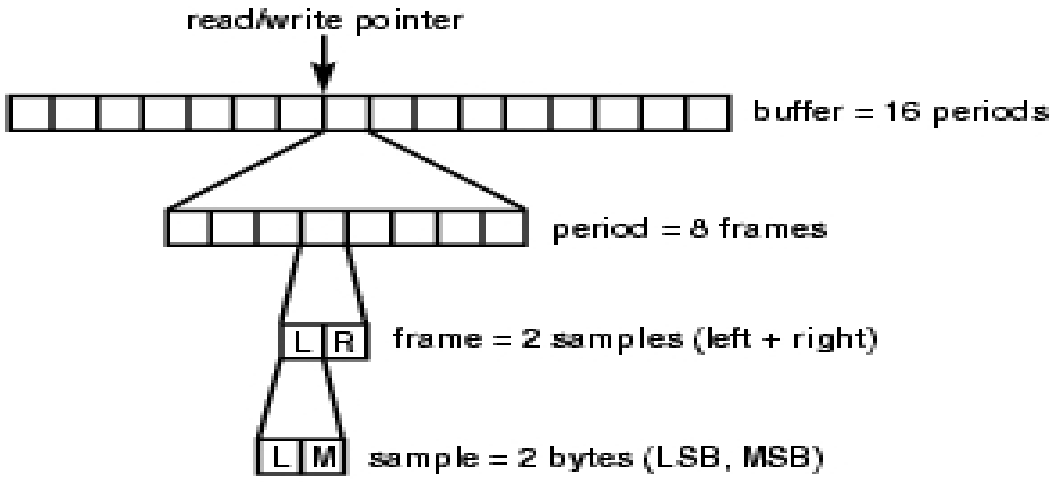


Figure 2.1: Application buffer

2.7 TERMINOLOGIES USED BY ALSA:

Many terminologies are used by ALSA:

- Capture
- Frame
- Playback
- PCM
- Full Duplex

2.7.1 CAPTURE:

Capture is stream types which receive data from the user. There is a difference between capture and recording. In recording it stores data somewhere instead of using ALSA's API.

2.7.2 FRAME:

A sample is a value on a single channel at a single point. When working with digital audio, in digital audio called channels includes single point on time. There is a collection of samples one per channel called frame.

2.7.3 PLAYBACK:

It is a stream type which delivers data to the world using speaker.

2.7.4 PCM:

PCM abbreviated as pulse code modulation which explain the method of representing analog signals in digital form. This method used by system audio interfaces and it is the basic part of ALSA API for audio programming.

2.7.5 FULL DUPLEX:

Full duplex is used for capture and play back on the same interface at the same time.

2.8 SETTING PARAMETER

There are types of parameters

- Hardware Parameter
- Software Parameter

2.8.1 HARDWARE PARAMETER

These are the main parameters which affects the hardware of audio interface.

- Sample Rate
- Sample Format
- Data Access

2.8.1.1 SAMPLE RATE:

Sample rate is used for controlling the rate at which A/D or D/A conversion is done. It also controls the flow of clock used to make digital audio data to or from outside the world.

2.8.1.2 SAMPLE FORMAT:

It controls the transferring of data to/from the interface.

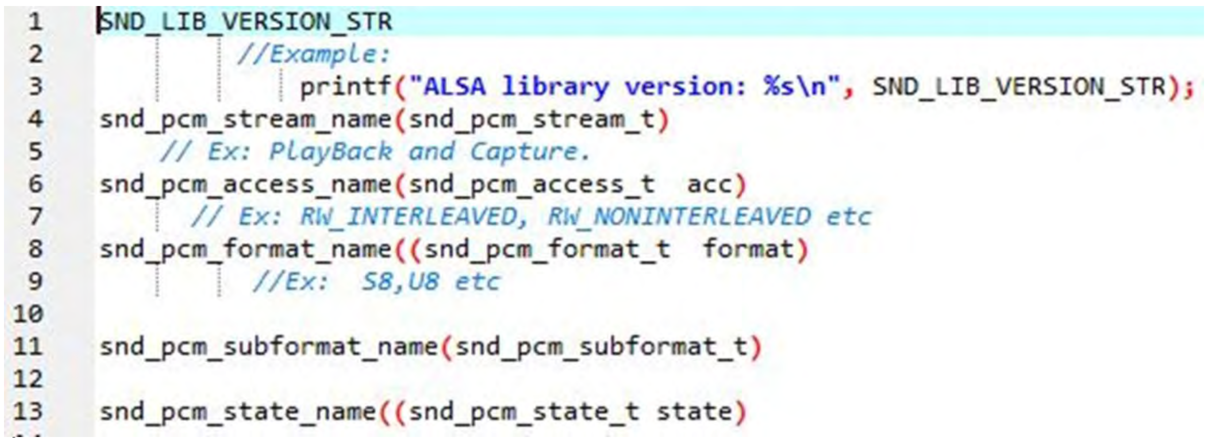
2.8.1.3 DATA ACCESS:

It controls the path in which the program will transmit/receive data from interface.

2.8.2 SOFTWARE PARAMETER

These function used to display some of the PCM data types and parameters use by ALSA. It is necessary to first include the header file which brings the definitions for all

ALSA library functions. In Figure 2.2, some of the most commonly used parameters are given which are used during the task



```
SND_LIB_VERSION_STR
        //Example:
            printf("ALSA library version: %s\n", SND_LIB_VERSION_STR);
snd_pcm_stream_name(snd_pcm_stream_t)
    // Ex: PlayBack and Capture.
snd_pcm_access_name(snd_pcm_access_t  acc)
      // Ex: RW_INTERLEAVED, RW_NONINTERLEAVED etc
snd_pcm_format_name((snd_pcm_format_t  format)
        //Ex:  S8,U8 etc

snd_pcm_subformat_name(snd_pcm_subformat_t)

snd_pcm_state_name((snd_pcm_state_t state)
```

Figure 2.2: Software parameters

9 TAKE A LOOK SOME PCM TYPES AND FORMATS

- **PCM STREAM TYPES :**
  1. Playback
  2. Capture

- **PCM ACCESS TYPES:**
  1. RW_INTERLEAVED
  2. RW_NON INTERLEAVED
- **PCM FORMATS:**
  1. S8 (signed 8 bits)
  2. U8 (unsigned 8 bits)
- **PCM SUB FORMATS:**
  1. STD (standard)
- **PCM STATES:**
  1. OPEN
  2. SETUP
  3. PREPARED
  4. RUNNING

2.10 DESIGNING OF SOUND APPLICATION:

Open interface:

Stream type ‘capture or playback’.

Setting hardware parameters (access mode, sample format, sample rate, channels etc.)

while there is data to be processed

Capture (read) PCM data or write (playback) PCM data.

Close interface.

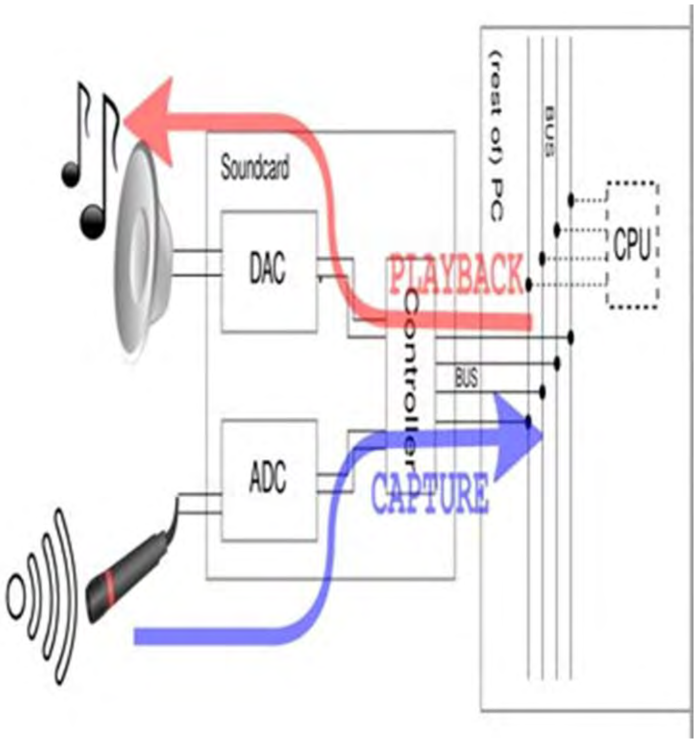


Figure 2.3: ALSA architecture

2.11 VOICE OVER IP

VoIP is used for the transmission of voice over internet instead of using switched telephone network. IP is used for data networking. Internet protocol adopted for noise networking using packetizing the data. The IP adopted for noise networking by forming the data into packets and transmit it as IP information packets. In modern era, VoIP is available on many android phones such as tablets which support internet access. VoIP is used for transmitting multimedia content also voice over IP networks. VoIP refers to deliver voice communications over wide area networks. VoIP is used for end to end communications. Application like soft phones and VoIP phones which runs VoIP over computer systems.

2.12 VOIP WORKING

VoIP uses codecs to encode audio into information packets and transmit those packets over IP networks and decode these packets back into information at the receiving end of the connection. For the elimination of circuit switched networks by using VoIP, infra-structure cost can be reduced. VoIP uses standard codecs for transmission like G.711 for decoding packets and G.729 for compressed packets. The quality of voice while using VoIP may suffer after compression but because of compression it reduces the requirement of bandwidth in real time. When voice is encoded onto IP network it uses RTP for transmission of data in real time.

2.13 APPLICATIONS OF VoIP

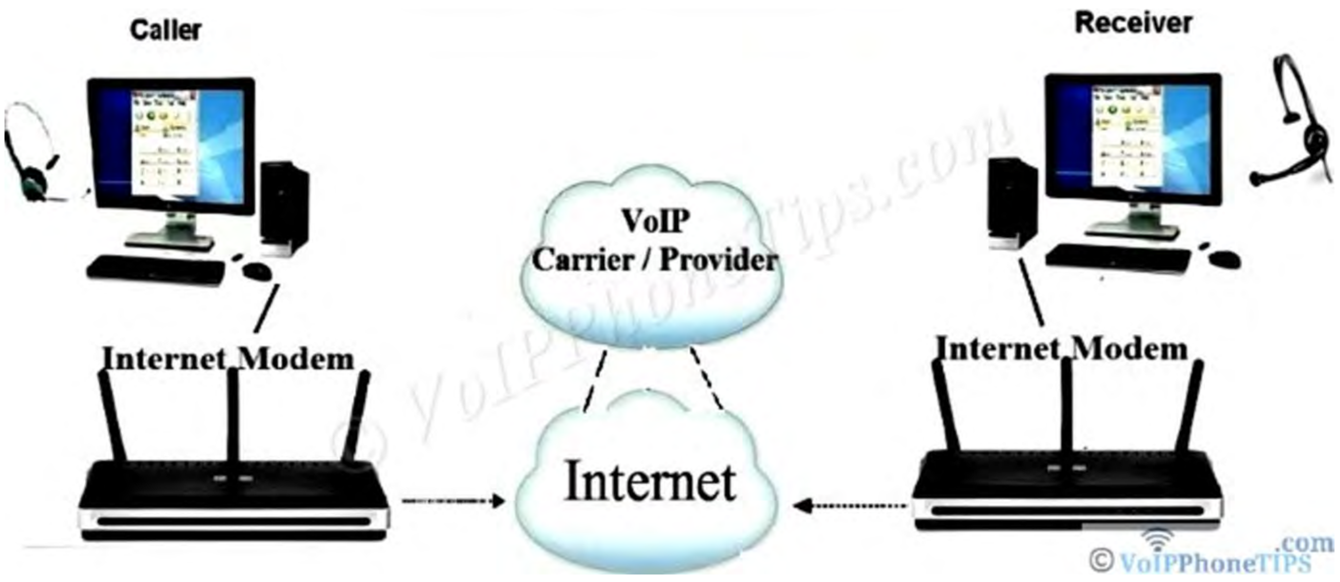


Figure 2.4: VoIP

VoIP is actually the process of transferring the voice data through the internet. VoIP allows making free calls to the other VoIP users. If speed of the internet is fast, VoIP technology helps to make calls around the world. Now a day’s to communicate with other people, VoIP plays a critical role in communication.

There are many application of VoIP in real time, some of them are

- send/receive voice mail

- call forwarding and call blocking
- collaboration of tools
- audio call conferences

In real time application VoIP uses UDP which requires lower delay, jitter and loss as compared to other data applications.

CHAPTER 3

LOW PRIORITY VoIP

3.1 OVERVIEW

Trx is a basic algorithm for the transmission of live audio from an operating system, Linux. It uses sound card or audio interface to send and receive programmed audio over IP networks.

It helps in point-to-point audio links or multi cast, like, radio station’s private transmitter links etc. Transmission dominates traditional streaming by sending high-grade broad audio having low-latency of about milliseconds and extremely quick recovery, like music.

3.2 INTERNET PROTOCOL

Internet protocol is an open source, based on the set of rules that can transmit the data over internet with minimum delay and the loss of content. IP has the ability that can automatically detect the path through internet with multiple paths. IP network requires that the entire user configured with the TCP/IP. One of the best common and largest IP network is internet. Each device has unique IP address, which is different from the other devices and helps for the transmission of data to the other device for communication. IP networks require all the devices must be configured with TCP/IP and all devices must have a valid IP network.

OPUS ENCODER:

Opus is used to sending the range audio applications like VoIP. It is used to escape the narrow band speech from low bit range rate to higher quality speech. It is also highly versatile audio codec. It provided some features which are given below

• Bit rates ranges from 6 kb/s to 510 kb/s.

• Sampling rates ranges from 8 kHz to 48 kHz.

• Frame sizes ranges from 2.5ms to 60ms.

• Supported for speech mono and stereo.

3.3 TRANSMISSION MODE

For the transmission of data between two devices using internet over IP network known as communication mode. Two different mode of communication are:

- Half duplex
- Full duplex

3.3.1 HALF DUPLEX

In half duplex transmission of data is unidirectional, as on one-way traffic. Only one device can transmit the audio at a time and the other device only receives the audio. Simply in this mode, one device can do only one task at a time. The main process of half duplex is given Figure 3.1

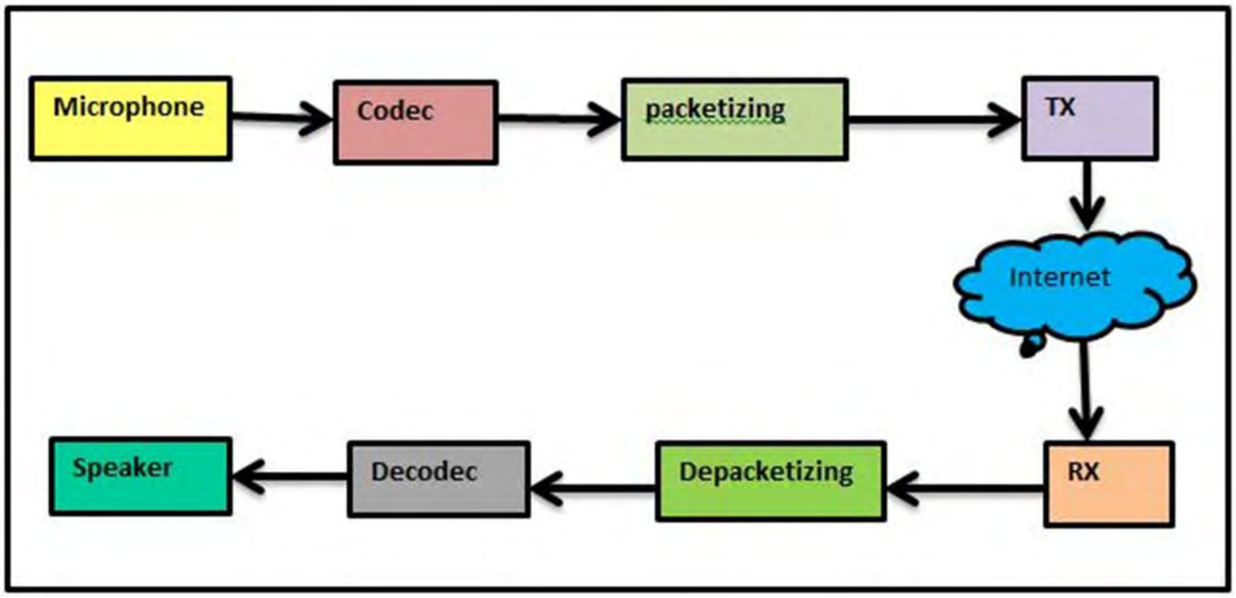
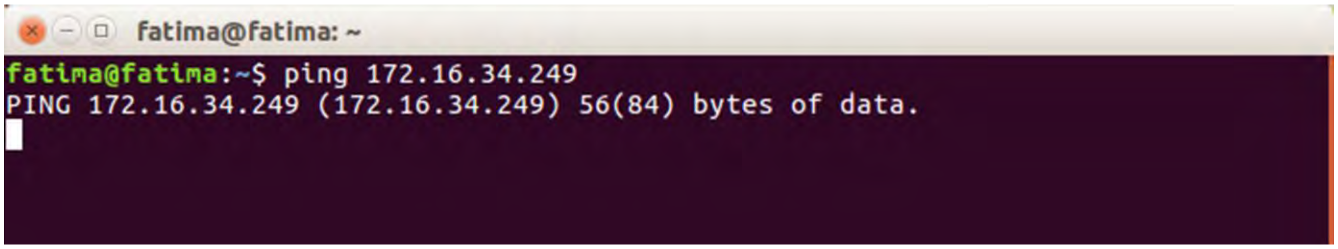


Figure 3.1: Half duplex

In this project, audio is transmitted in half duplex mode. The whole process of the transmitting real time audio is explained below

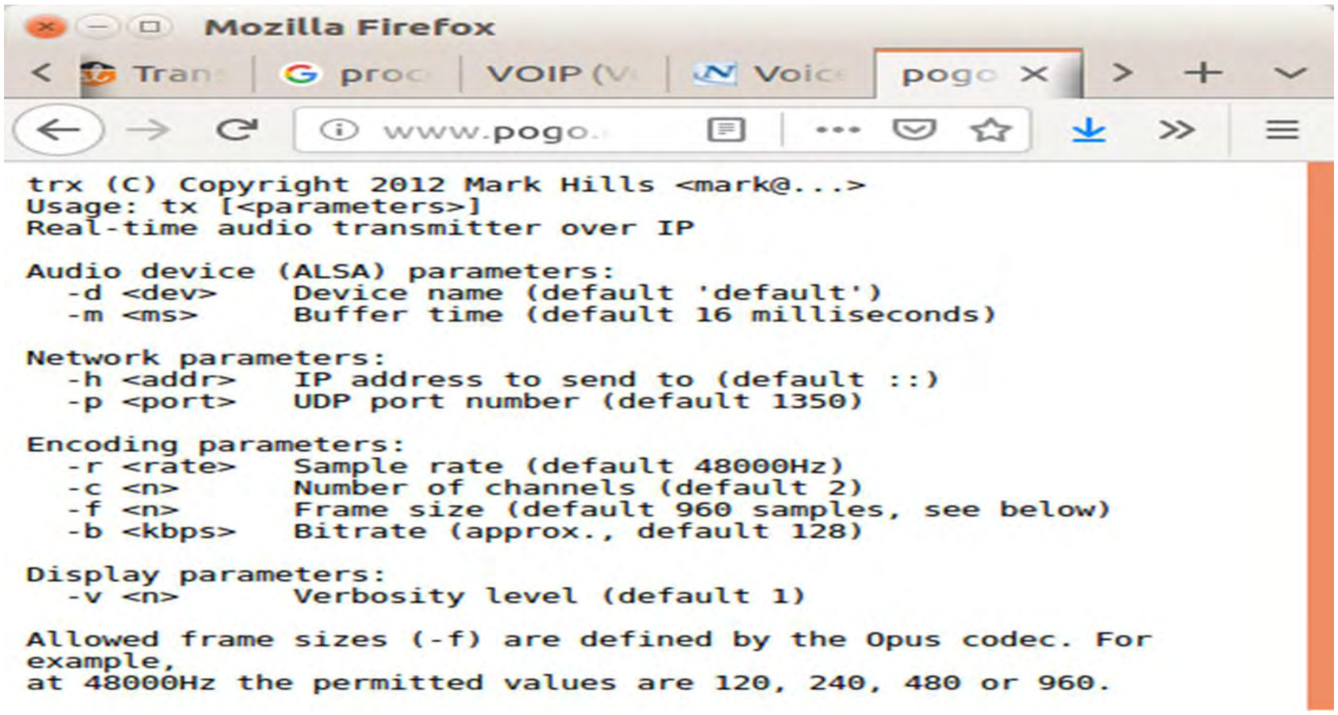
For the transmission of audio, firstly both the devices are connected with each other over internet. Both the devices are connected with each by using the command as given in Figure3.2 below



```
fatima@fatima:~$ ping 172.16.34.249
PING 172.16.34.249 (172.16.34.249) 56(84) bytes of data.
```

Figure 3.2: Ping command

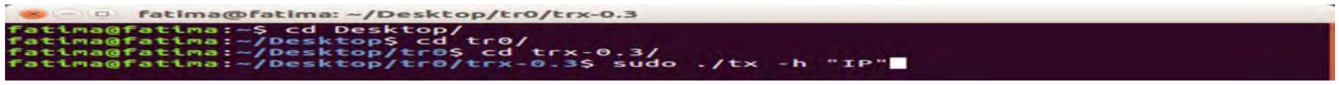
If the bytes of data are transmitted it means both the devices are connected with each other through internet. If both the devices are not connected with each other, data cannot be transmitted. To resolve this problem check the internet connection. When the devices are connected then talk in microphone through the device which transmit the audio. The audio signal is then passed through the codecs. Codecs is actually the process in which the audio converted into data form. Codecs is the main process which allows us to transmit the audio in real time to other device or device. Many of the sound samples are taken in by codecs within a few seconds. These samples are used for hearing clear voice during conferences or phone calls. If the codecs not runs properly due to slow internet the voice on the other end are not cleared. Once the codecs process is done, the data which is obtained from codecs, that data then divided into many small pieces called packets. These packets are transmitted to the other end in just few milliseconds before reassembled in proper order. Each packet take a different path to reach at the destination, the process of sending the packets is called packet switching. It is necessary that both the codecs and the packet switching work properly. If both of these are working properly then the sound transmitted, different parameters are used for the transmission of voice over IP network which are given in Figure 3.3.



```
trx (C) Copyright 2012 Mark Hills <mark@...>
Usage: tx [<parameters>]
Real-time audio transmitter over IP

Audio device (ALSA) parameters:
  -d <dev>    Device name (default 'default')
  -m <ms>     Buffer time (default 16 milliseconds)

Network parameters:
  -h <addr>   IP address to send to (default ::)
  -p <port>   UDP port number (default 1350)

Encoding parameters:
  -r <rate>   Sample rate (default 48000Hz)
  -c <n>      Number of channels (default 2)
  -f <n>      Frame size (default 960 samples, see below)
  -b <kbps>   Bitrate (approx., default 128)

Display parameters:
  -v <n>      Verbosity level (default 1)

Allowed frame sizes (-f) are defined by the Opus codec. For
example,
at 48000Hz the permitted values are 120, 240, 480 or 960.
```

Figure 3.3: Transmitter parameter

To transmit the audio from one end, sender has to mention the IP of the receiving end. If the IP network is not mention from the transmitted end then audio is received in the same device, default IP is used, from where the audio is transmitted. To transmit the audio, sampling rate and the buffer size are most important parameters. For the transmission of audio, sampling rate is 16000 measured in Hz and the buffer size is 16 which are measured in ms.

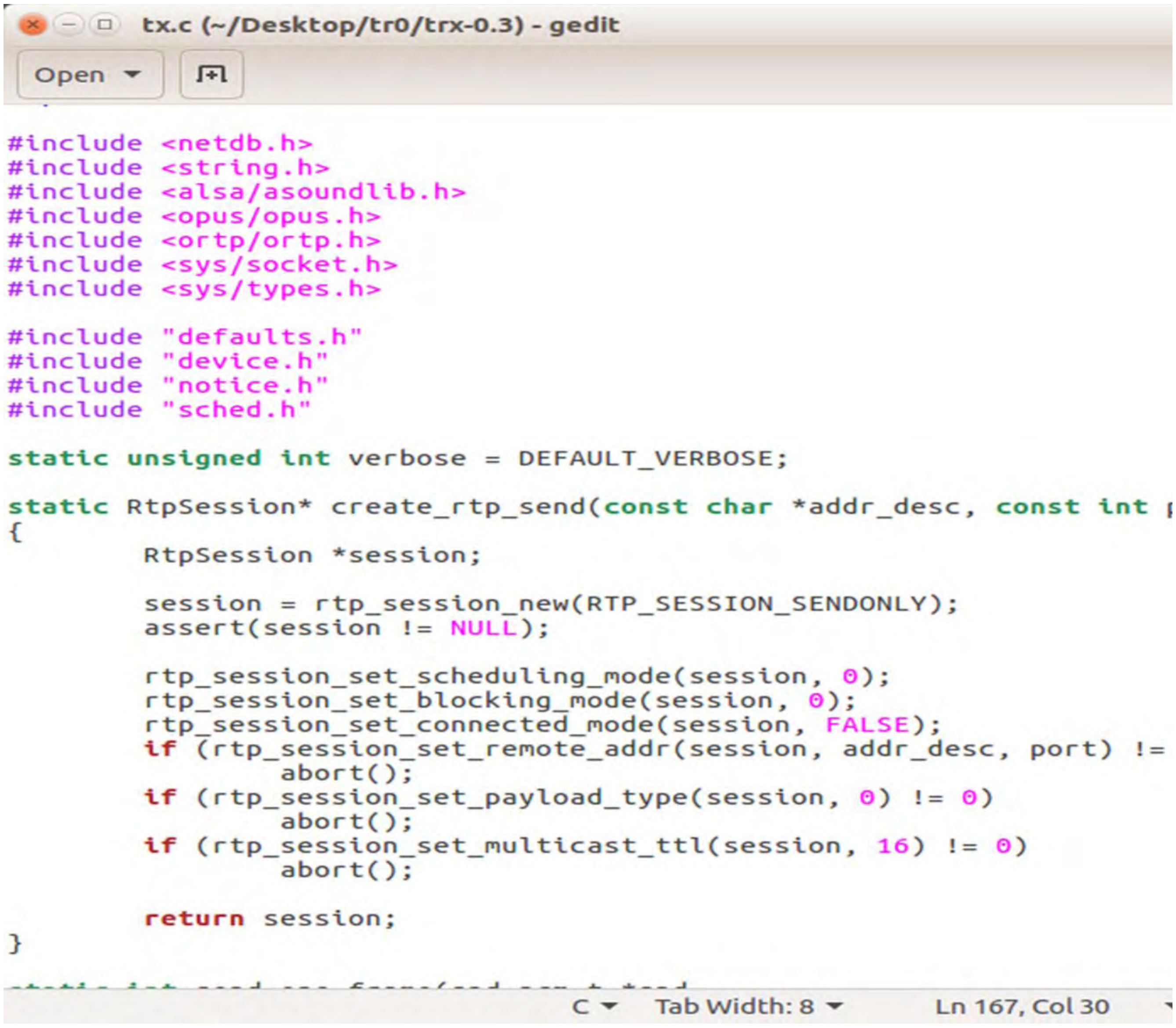


```
fatima@fatima:~$ cd Desktop/
fatima@fatima:~/Desktop$ cd tr0/
fatima@fatima:~/Desktop/tr0$ cd trx-0.3/
fatima@fatima:~/Desktop/tr0/trx-0.3$ sudo ./tx -h "IP"
```

Figure 3.4: How to run transmission code

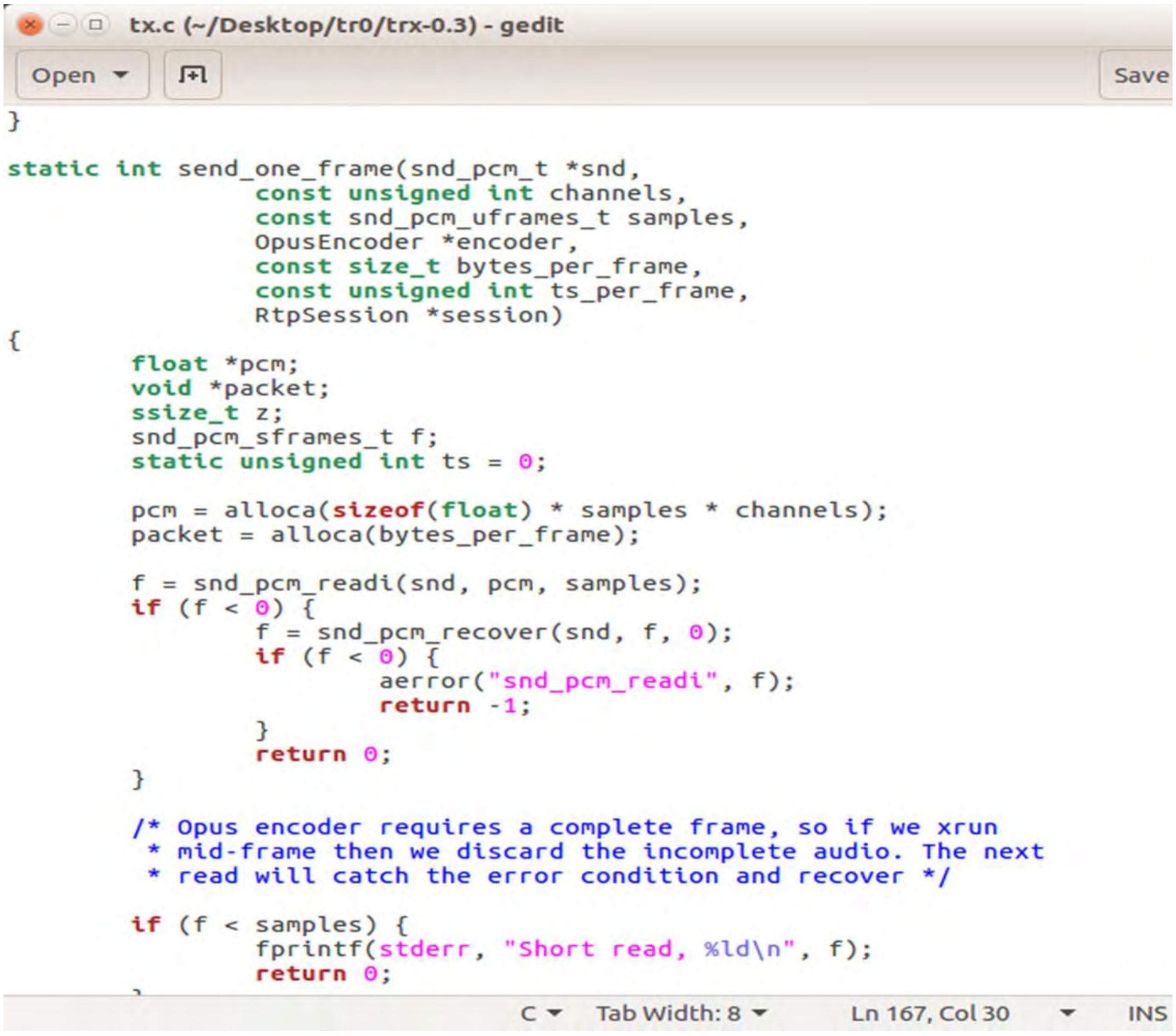
If the buffer size is not accurate, the codecs process not done properly. To transmit the audio from the device, firstly check that the devices are connected and also user has to compile the transmission code. Figure 3.4 shows how the transmission code works.

The code to transmit the audio is given below in Figure 3.5.



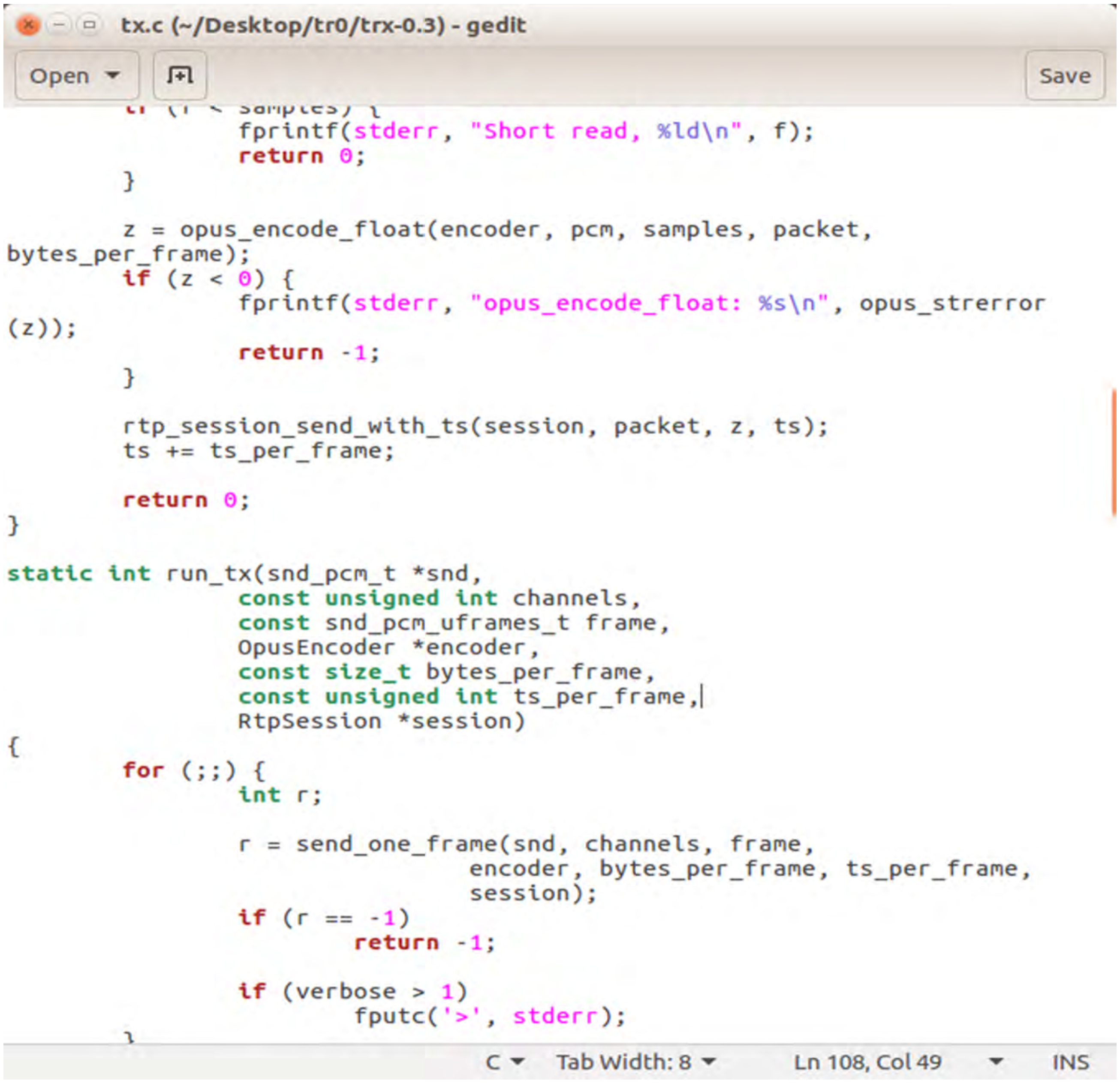
```
#include <netdb.h>
#include <string.h>
#include <alsa/asoundlib.h>
#include <opus/opus.h>
#include <ortp/ortp.h>
#include <sys/socket.h>
#include <sys/types.h>

#include "defaults.h"
#include "device.h"
#include "notice.h"
#include "sched.h"

static unsigned int verbose = DEFAULT_VERBOSE;

static RtpSession* create_rtp_send(const char *addr_desc, const int
{
        RtpSession *session;

        session = rtp_session_new(RTP_SESSION_SENDONLY);
        assert(session != NULL);

        rtp_session_set_scheduling_mode(session, 0);
        rtp_session_set_blocking_mode(session, 0);
        rtp_session_set_connected_mode(session, FALSE);
        if (rtp_session_set_remote_addr(session, addr_desc, port) !=
                abort();
        if (rtp_session_set_payload_type(session, 0) != 0)
                abort();
        if (rtp_session_set_multicast_ttl(session, 16) != 0)
                abort();

        return session;
}
```

Figure 3.5: Trx(a)



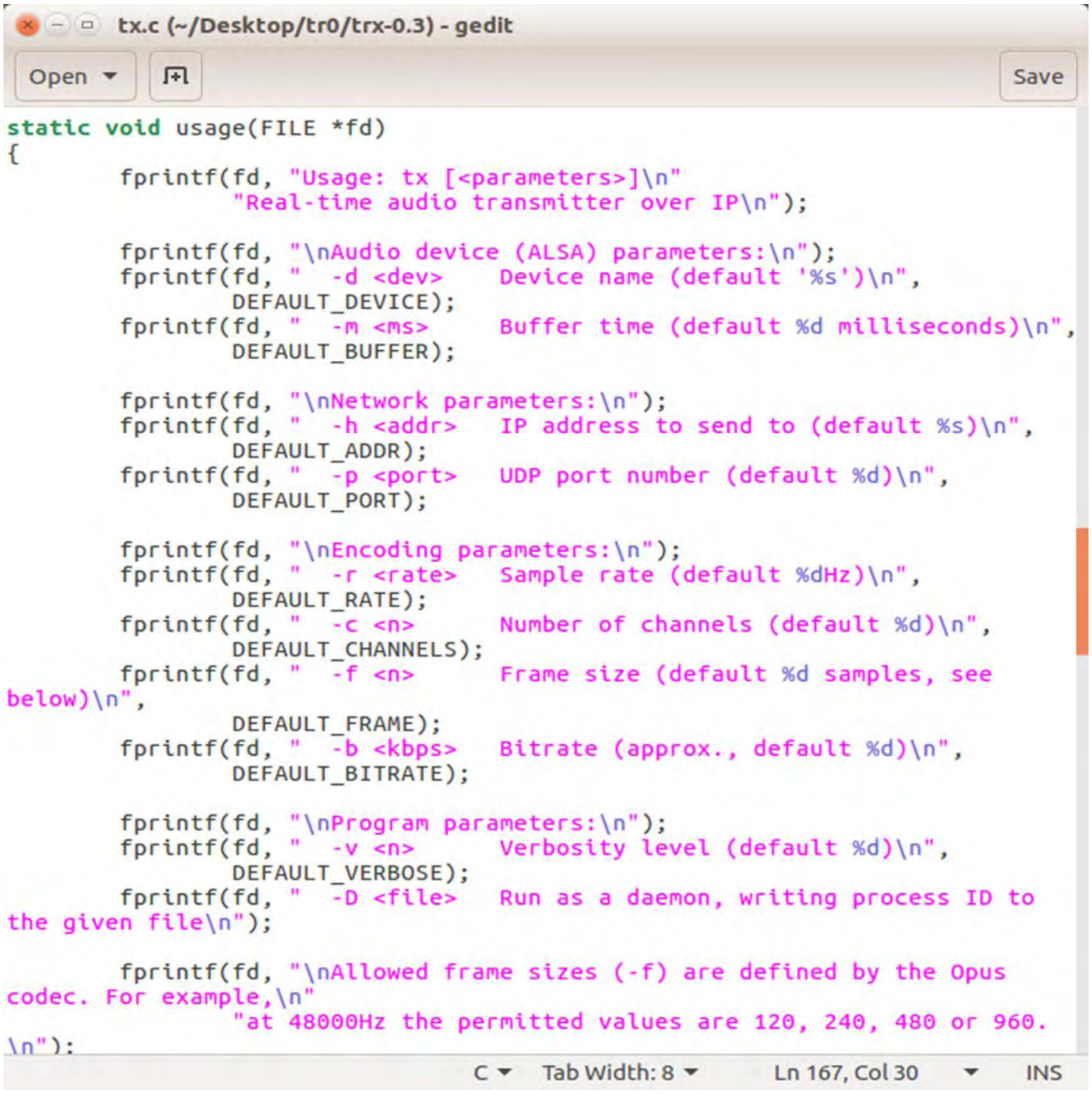
```
}

static int send_one_frame(snd_pcm_t *snd,
                const unsigned int channels,
                const snd_pcm_uframes_t samples,
                OpusEncoder *encoder,
                const size_t bytes_per_frame,
                const unsigned int ts_per_frame,
                RtpSession *session)
{
        float *pcm;
        void *packet;
        ssize_t z;
        snd_pcm_sframes_t f;
        static unsigned int ts = 0;

        pcm = alloca(sizeof(float) * samples * channels);
        packet = alloca(bytes_per_frame);

        f = snd_pcm_readi(snd, pcm, samples);
        if (f < 0) {
                f = snd_pcm_recover(snd, f, 0);
                if (f < 0) {
                        aerror("snd_pcm_readi", f);
                        return -1;
                }
                return 0;
        }

        /* Opus encoder requires a complete frame, so if we xrun
         * mid-frame then we discard the incomplete audio. The next
         * read will catch the error condition and recover */

        if (f < samples) {
                fprintf(stderr, "Short read, %ld\n", f);
                return 0;
```

Figure 3.5: Trx (b)



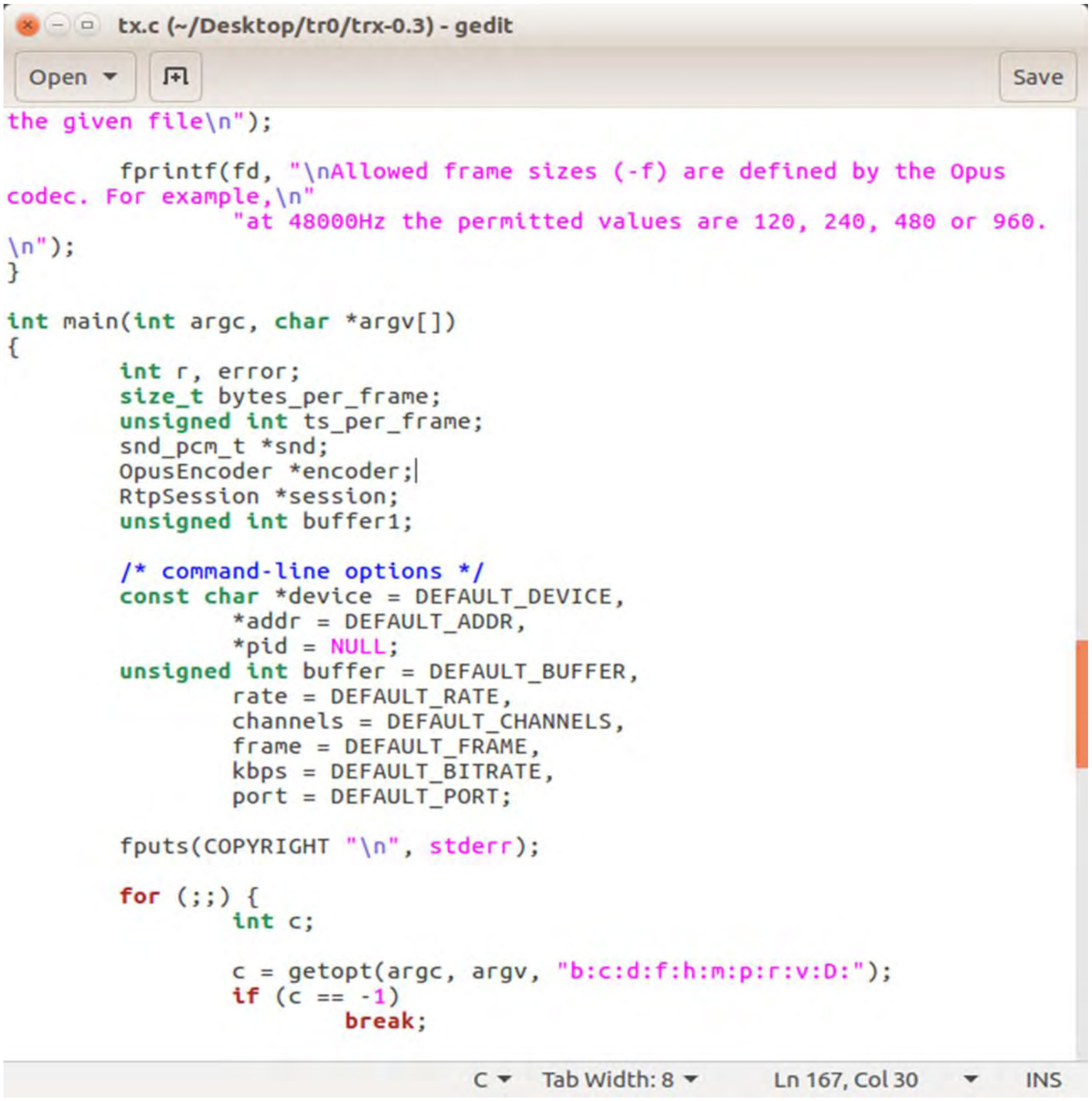
```
                fprintf(stderr, "Short read, %ld\n", f);
                return 0;
        }

        z = opus_encode_float(encoder, pcm, samples, packet,
bytes_per_frame);
        if (z < 0) {
                fprintf(stderr, "opus_encode_float: %s\n", opus_strerror
(z));
                return -1;
        }

        rtp_session_send_with_ts(session, packet, z, ts);
        ts += ts_per_frame;

        return 0;
}

static int run_tx(snd_pcm_t *snd,
                const unsigned int channels,
                const snd_pcm_uframes_t frame,
                OpusEncoder *encoder,
                const size_t bytes_per_frame,
                const unsigned int ts_per_frame,
                RtpSession *session)
{
        for (;;) {
                int r;

                r = send_one_frame(snd, channels, frame,
                                encoder, bytes_per_frame, ts_per_frame,
                                session);
                if (r == -1)
                        return -1;

                if (verbose > 1)
                        fputc('>', stderr);
```

Figure 3.5: Trx (c)



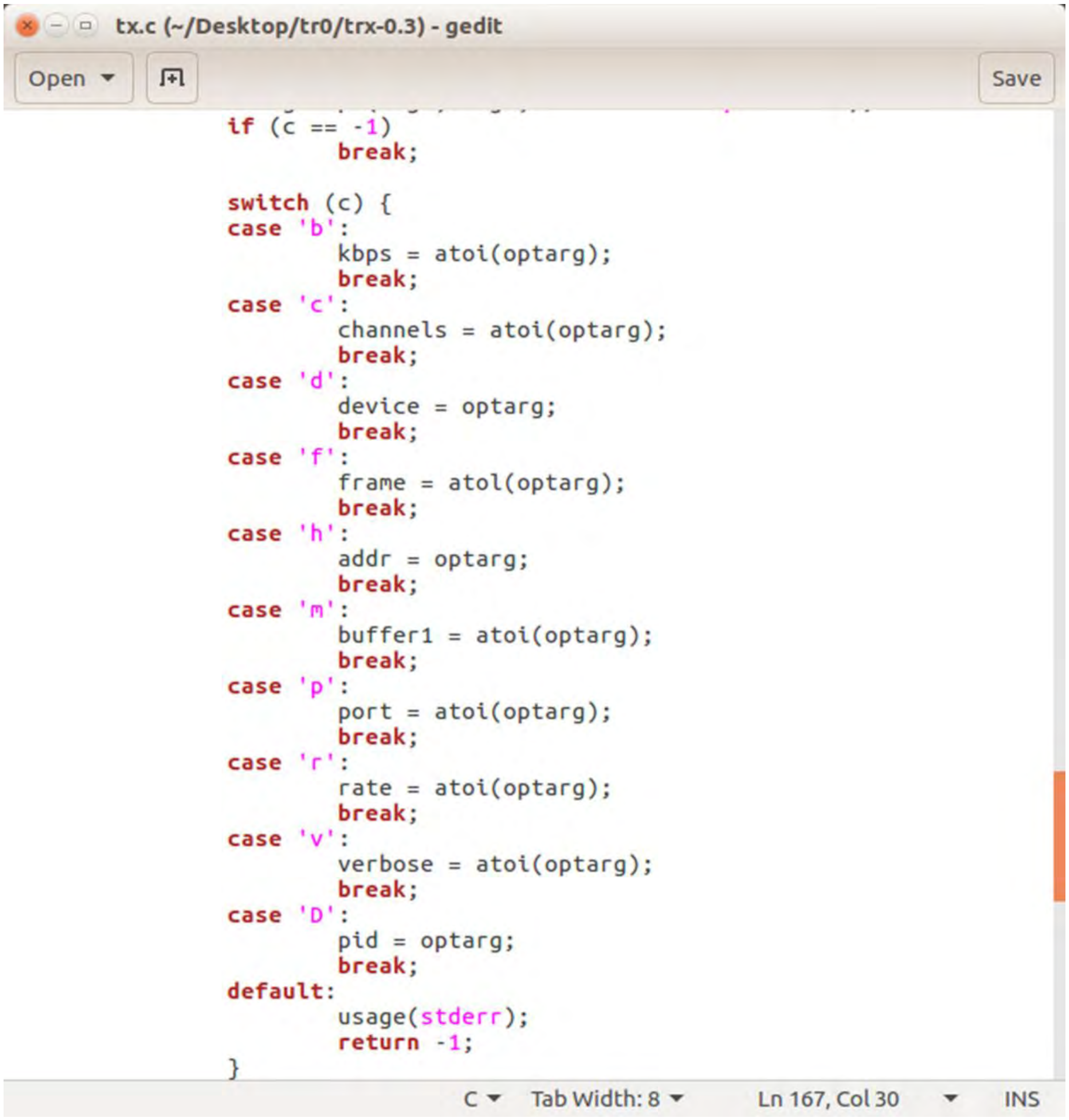
```
static void usage(FILE *fd)
{
        fprintf(fd, "Usage: tx [<parameters>]\n"
                "Real-time audio transmitter over IP\n");

        fprintf(fd, "\nAudio device (ALSA) parameters:\n");
        fprintf(fd, "  -d <dev>    Device name (default '%s')\n",
                DEFAULT_DEVICE);
        fprintf(fd, "  -m <ms>     Buffer time (default %d milliseconds)\n",
                DEFAULT_BUFFER);

        fprintf(fd, "\nNetwork parameters:\n");
        fprintf(fd, "  -h <addr>   IP address to send to (default %s)\n",
                DEFAULT_ADDR);
        fprintf(fd, "  -p <port>   UDP port number (default %d)\n",
                DEFAULT_PORT);

        fprintf(fd, "\nEncoding parameters:\n");
        fprintf(fd, "  -r <rate>   Sample rate (default %dHz)\n",
                DEFAULT_RATE);
        fprintf(fd, "  -c <n>      Number of channels (default %d)\n",
                DEFAULT_CHANNELS);
        fprintf(fd, "  -f <n>      Frame size (default %d samples, see
below)\n",
                DEFAULT_FRAME);
        fprintf(fd, "  -b <kbps>   Bitrate (approx., default %d)\n",
                DEFAULT_BITRATE);

        fprintf(fd, "\nProgram parameters:\n");
        fprintf(fd, "  -v <n>      Verbosity level (default %d)\n",
                DEFAULT_VERBOSE);
        fprintf(fd, "  -D <file>   Run as a daemon, writing process ID to
the given file\n");

        fprintf(fd, "\nAllowed frame sizes (-f) are defined by the Opus
codec. For example,\n"
                "at 48000Hz the permitted values are 120, 240, 480 or 960.
\n");
```

Figure 3.5: Trx (d)



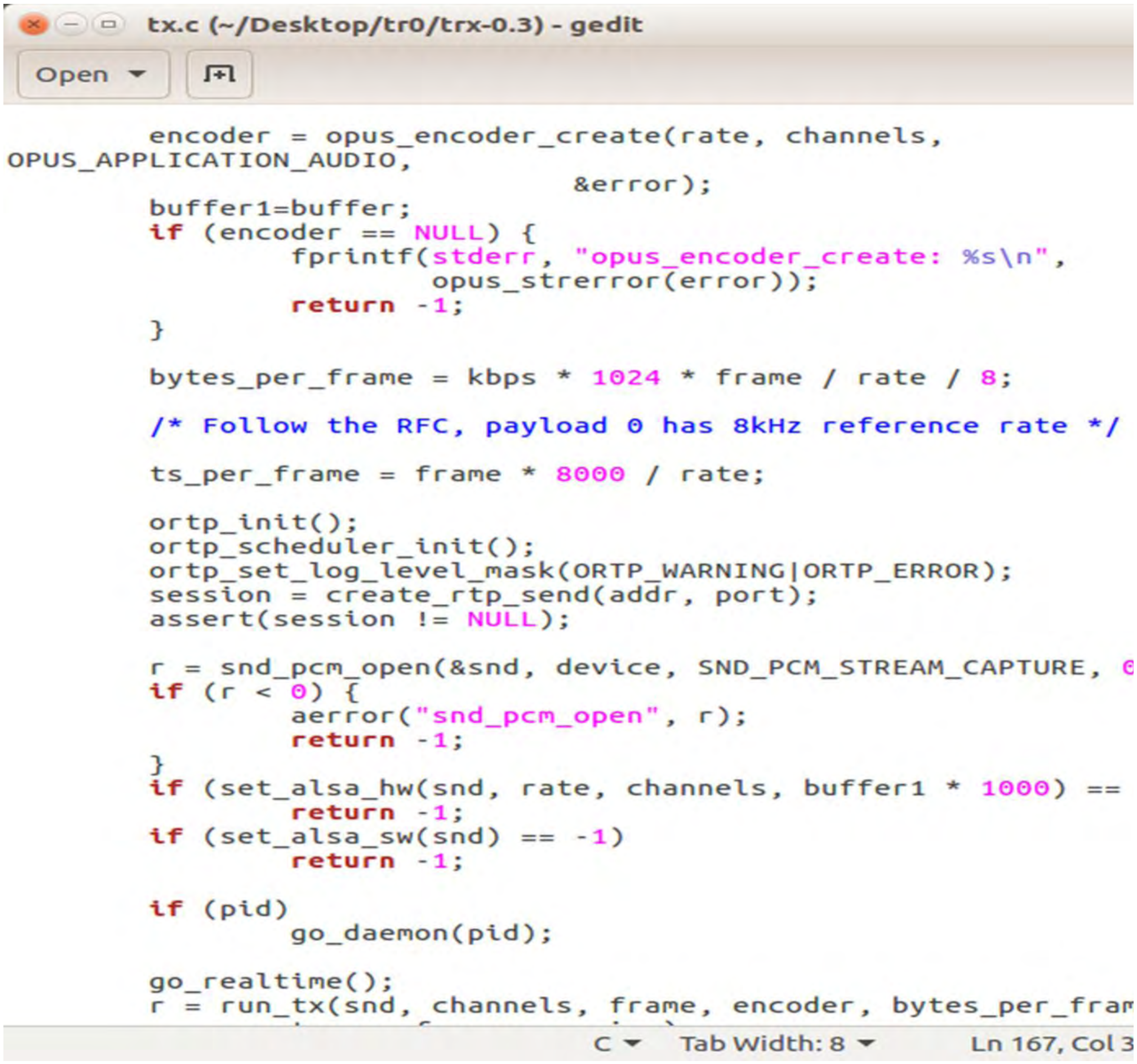
```
the given file\n");

        fprintf(fd, "\nAllowed frame sizes (-f) are defined by the Opus
codec. For example,\n"
                "at 48000Hz the permitted values are 120, 240, 480 or 960.
\n");
}

int main(int argc, char *argv[])
{
        int r, error;
        size_t bytes_per_frame;
        unsigned int ts_per_frame;
        snd_pcm_t *snd;
        OpusEncoder *encoder;
        RtpSession *session;
        unsigned int buffer1;

        /* command-line options */
        const char *device = DEFAULT_DEVICE,
                *addr = DEFAULT_ADDR,
                *pid = NULL;
        unsigned int buffer = DEFAULT_BUFFER,
                rate = DEFAULT_RATE,
                channels = DEFAULT_CHANNELS,
                frame = DEFAULT_FRAME,
                kbps = DEFAULT_BITRATE,
                port = DEFAULT_PORT;

        fputs(COPYRIGHT "\n", stderr);

        for (;;) {
                int c;

                c = getopt(argc, argv, "b:c:d:f:h:m:p:r:v:D:");
                if (c == -1)
                        break;
```

Figure 3.5: Trx (e)



```
        if (c == -1)
                break;

        switch (c) {
        case 'b':
                kbps = atoi(optarg);
                break;
        case 'c':
                channels = atoi(optarg);
                break;
        case 'd':
                device = optarg;
                break;
        case 'f':
                frame = atol(optarg);
                break;
        case 'h':
                addr = optarg;
                break;
        case 'm':
                buffer1 = atoi(optarg);
                break;
        case 'p':
                port = atoi(optarg);
                break;
        case 'r':
                rate = atoi(optarg);
                break;
        case 'v':
                verbose = atoi(optarg);
                break;
        case 'D':
                pid = optarg;
                break;
        default:
                usage(stderr);
                return -1;
        }
```

Figure 3.5: Trx (f)

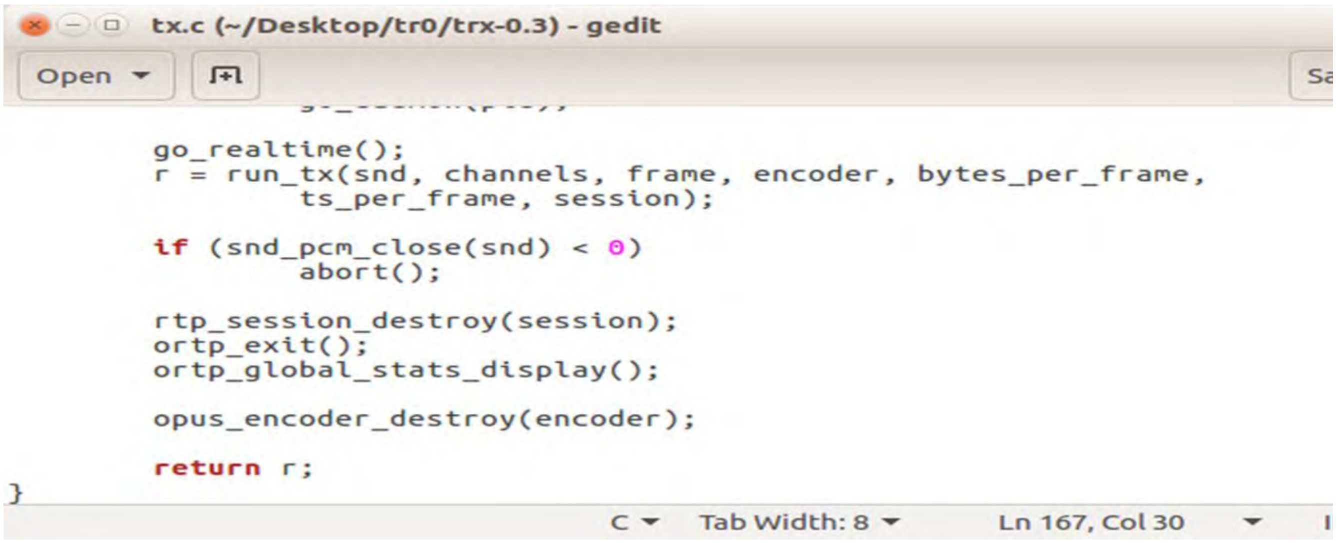


tx.c (~/Desktop/tr0/trx-0.3) - gedit

```
        encoder = opus_encoder_create(rate, channels,
OPUS_APPLICATION_AUDIO,
                                      &error);
        buffer1=buffer;
        if (encoder == NULL) {
                fprintf(stderr, "opus_encoder_create: %s\n",
                        opus_strerror(error));
                return -1;
        }

        bytes_per_frame = kbps * 1024 * frame / rate / 8;

        /* Follow the RFC, payload 0 has 8kHz reference rate */

        ts_per_frame = frame * 8000 / rate;

        ortp_init();
        ortp_scheduler_init();
        ortp_set_log_level_mask(ORTP_WARNING|ORTP_ERROR);
        session = create_rtp_send(addr, port);
        assert(session != NULL);

        r = snd_pcm_open(&snd, device, SND_PCM_STREAM_CAPTURE, 0
        if (r < 0) {
                aerror("snd_pcm_open", r);
                return -1;
        }
        if (set_alsa_hw(snd, rate, channels, buffer1 * 1000) ==
                return -1;
        if (set_alsa_sw(snd) == -1)
                return -1;

        if (pid)
                go_daemon(pid);

        go_realtime();
        r = run_tx(snd, channels, frame, encoder, bytes_per_fram
```

C Tab Width: 8 Ln 167, Col 3

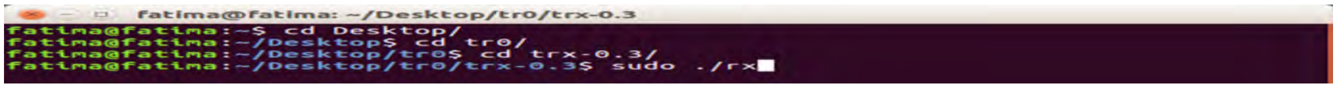
Figure 3.5: Trx (g)



```
        go_realtime();
        r = run_tx(snd, channels, frame, encoder, bytes_per_frame,
                ts_per_frame, session);

        if (snd_pcm_close(snd) < 0)
                abort();

        rtp_session_destroy(session);
        ortp_exit();
        ortp_global_stats_display();

        opus_encoder_destroy(encoder);

        return r;
}
```

Figure 3.5: Trx (h)

At the receiver end, first the de-packetizing is done. After this the data which are transmitted are converted into the signal form in real time by using D/A converter known as decodec process. The receiver does not need to mention IP network of the transmitter end. To run the receiver end code, to receive audio the command used is given below in Figure 3.5.

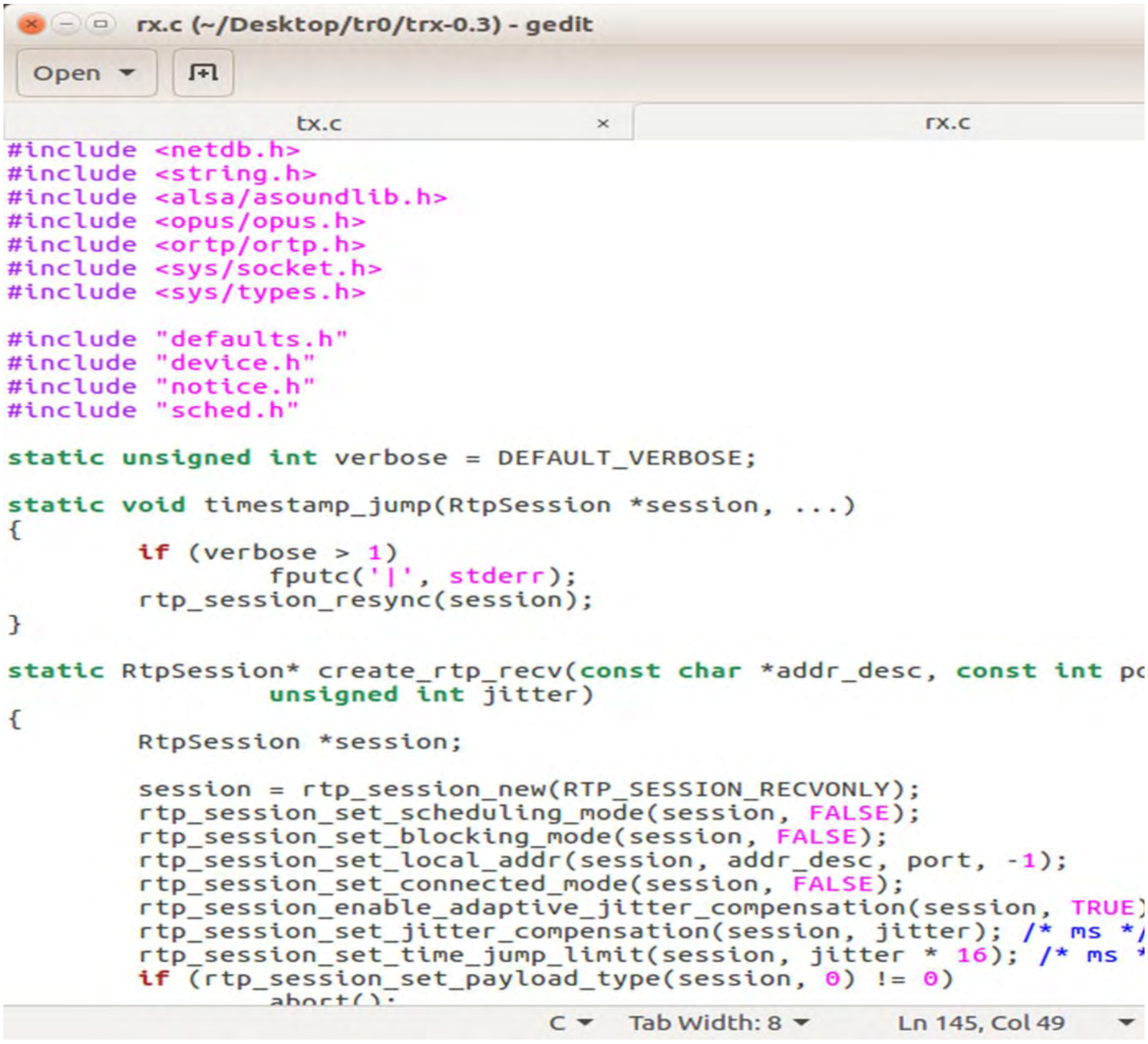


```
fatima@fatima:~$ cd Desktop/
fatima@fatima:~/Desktop$ cd tr0/
fatima@fatima:~/Desktop/tr0$ cd trx-0.3/
fatima@fatima:~/Desktop/tr0/trx-0.3$ sudo ./rx
```

Figure 3.5: Receiving command

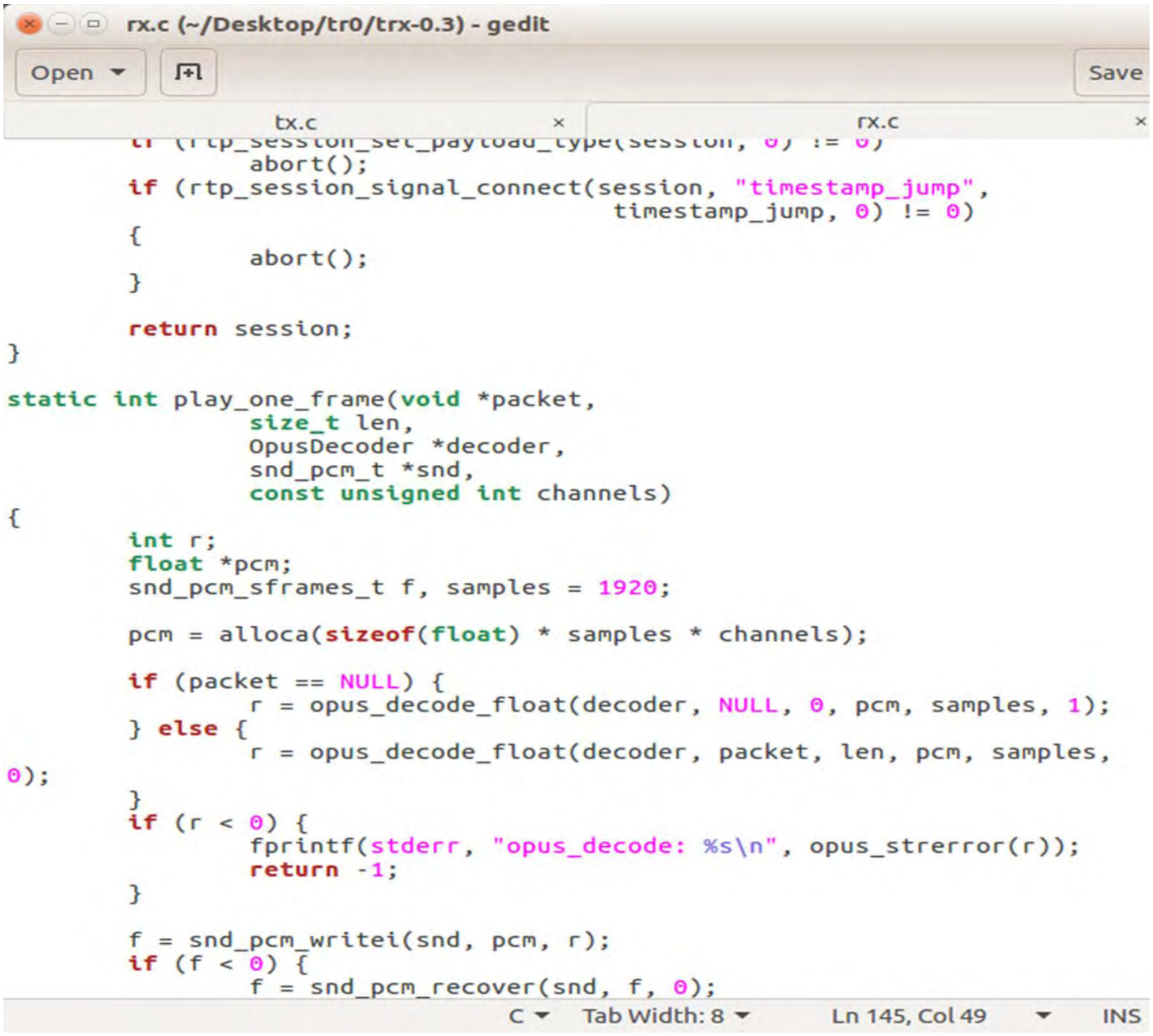
For receiving the audio, same parameter which are used for transmission instead of only jitter buffer, size of jitter buffer is more important which is equal to the size of buffer. Jitter buffer is used to store the incoming packets of data. Jitter buffer temporary stores the data packets. The code to receive the audio is given in Figure 3.7.

The code to receive the audio is given in Figure 3.7.



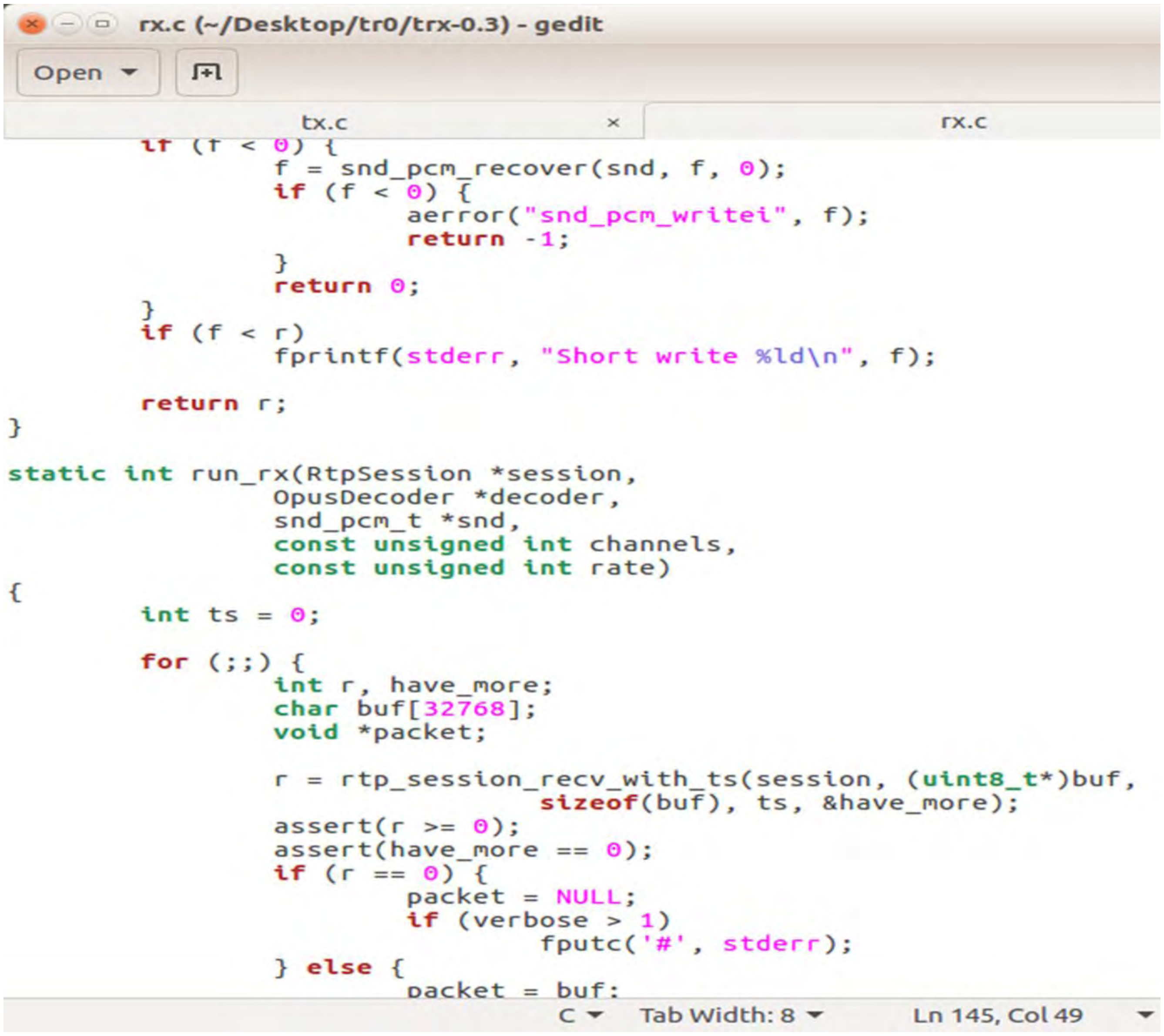
```
#include <netdb.h>
#include <string.h>
#include <alsa/asoundlib.h>
#include <opus/opus.h>
#include <ortp/ortp.h>
#include <sys/socket.h>
#include <sys/types.h>

#include "defaults.h"
#include "device.h"
#include "notice.h"
#include "sched.h"

static unsigned int verbose = DEFAULT_VERBOSE;

static void timestamp_jump(RtpSession *session, ...)
{
        if (verbose > 1)
                fputc('|', stderr);
        rtp_session_resync(session);
}

static RtpSession* create_rtp_recv(const char *addr_desc, const int po
                unsigned int jitter)
{
        RtpSession *session;

        session = rtp_session_new(RTP_SESSION_RECVONLY);
        rtp_session_set_scheduling_mode(session, FALSE);
        rtp_session_set_blocking_mode(session, FALSE);
        rtp_session_set_local_addr(session, addr_desc, port, -1);
        rtp_session_set_connected_mode(session, FALSE);
        rtp_session_enable_adaptive_jitter_compensation(session, TRUE)
        rtp_session_set_jitter_compensation(session, jitter); /* ms */
        rtp_session_set_time_jump_limit(session, jitter * 16); /* ms *
        if (rtp_session_set_payload_type(session, 0) != 0)
                abort();
```

Figure 3.7: Rx(a)



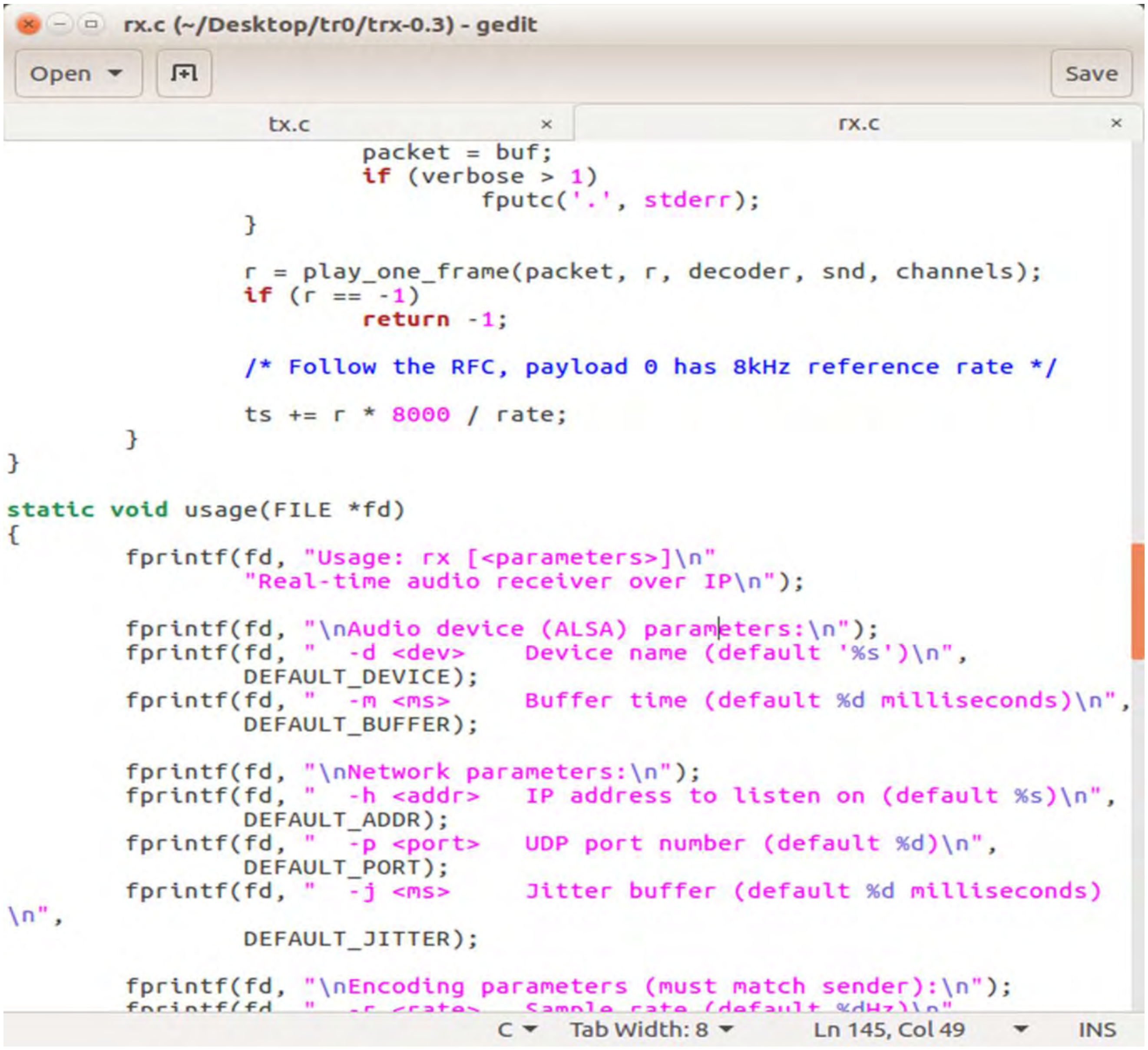
```
                abort();
        if (rtp_session_signal_connect(session, "timestamp_jump",
                                              timestamp_jump, 0) != 0)
        {
                abort();
        }

        return session;
}

static int play_one_frame(void *packet,
                size_t len,
                OpusDecoder *decoder,
                snd_pcm_t *snd,
                const unsigned int channels)
{
        int r;
        float *pcm;
        snd_pcm_sframes_t f, samples = 1920;

        pcm = alloca(sizeof(float) * samples * channels);

        if (packet == NULL) {
                r = opus_decode_float(decoder, NULL, 0, pcm, samples, 1);
        } else {
                r = opus_decode_float(decoder, packet, len, pcm, samples,
0);
        }
        if (r < 0) {
                fprintf(stderr, "opus_decode: %s\n", opus_strerror(r));
                return -1;
        }

        f = snd_pcm_writei(snd, pcm, r);
        if (f < 0) {
                f = snd_pcm_recover(snd, f, 0);
```

Figure 3.7: Rx (b)


rx.c (~/Desktop/tr0/trx-0.3) - gedit

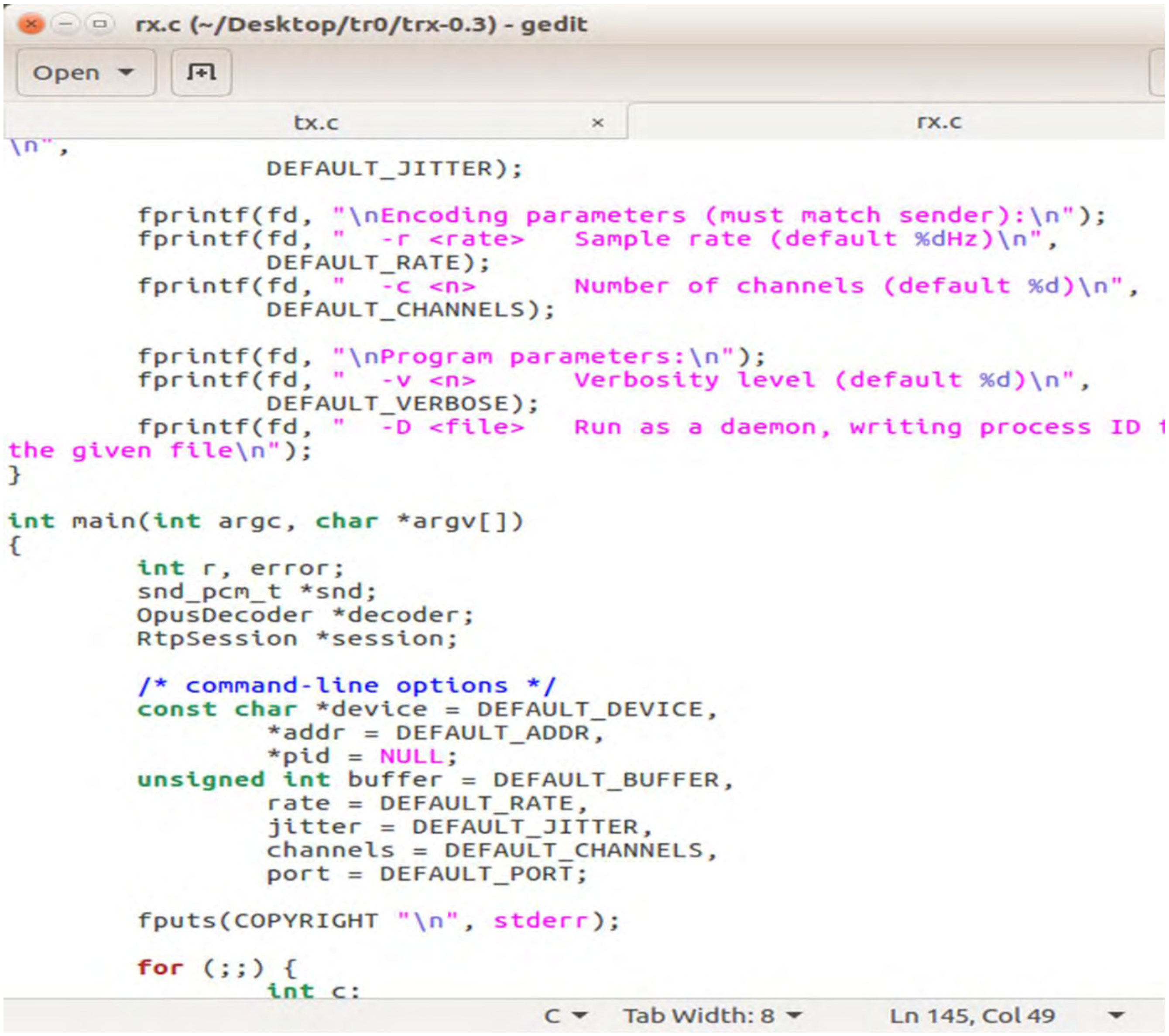
```
        if (f < 0) {
                f = snd_pcm_recover(snd, f, 0);
                if (f < 0) {
                        aerror("snd_pcm_writei", f);
                        return -1;
                }
                return 0;
        }
        if (f < r)
                fprintf(stderr, "Short write %ld\n", f);

        return r;
}

static int run_rx(RtpSession *session,
                OpusDecoder *decoder,
                snd_pcm_t *snd,
                const unsigned int channels,
                const unsigned int rate)
{
        int ts = 0;

        for (;;) {
                int r, have_more;
                char buf[32768];
                void *packet;

                r = rtp_session_recv_with_ts(session, (uint8_t*)buf,
                                sizeof(buf), ts, &have_more);
                assert(r >= 0);
                assert(have_more == 0);
                if (r == 0) {
                        packet = NULL;
                        if (verbose > 1)
                                fputc('#', stderr);
                } else {
                        packet = buf;
```

Figure 3.7: Rx(c)



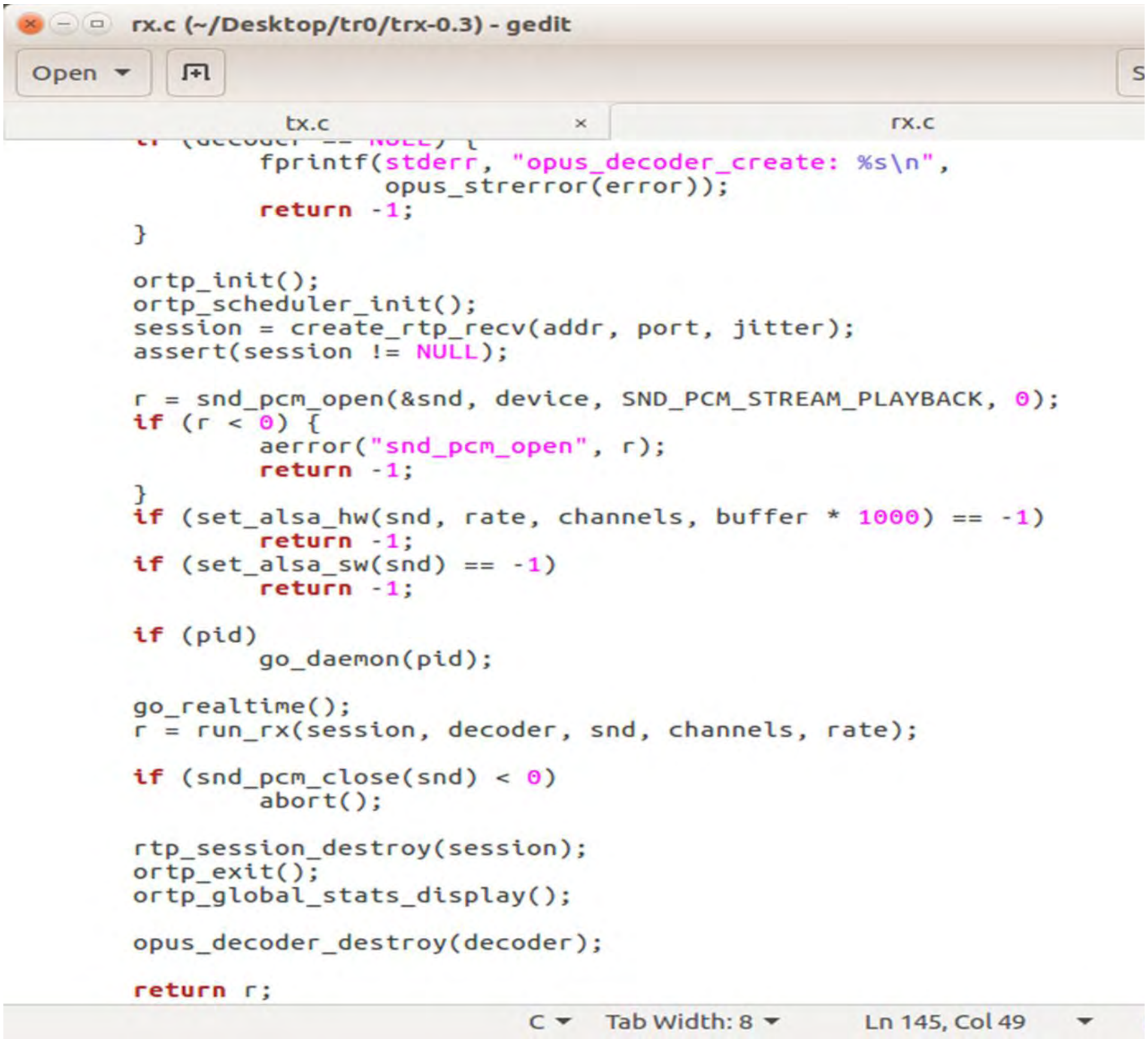
```
                        packet = buf;
                        if (verbose > 1)
                                fputc('.', stderr);
                }

                r = play_one_frame(packet, r, decoder, snd, channels);
                if (r == -1)
                        return -1;

                /* Follow the RFC, payload 0 has 8kHz reference rate */

                ts += r * 8000 / rate;
        }
}

static void usage(FILE *fd)
{
        fprintf(fd, "Usage: rx [<parameters>]\n"
                "Real-time audio receiver over IP\n");

        fprintf(fd, "\nAudio device (ALSA) parameters:\n");
        fprintf(fd, "  -d <dev>    Device name (default '%s')\n",
                DEFAULT_DEVICE);
        fprintf(fd, "  -m <ms>     Buffer time (default %d milliseconds)\n",
                DEFAULT_BUFFER);

        fprintf(fd, "\nNetwork parameters:\n");
        fprintf(fd, "  -h <addr>   IP address to listen on (default %s)\n",
                DEFAULT_ADDR);
        fprintf(fd, "  -p <port>   UDP port number (default %d)\n",
                DEFAULT_PORT);
        fprintf(fd, "  -j <ms>     Jitter buffer (default %d milliseconds)
\n",
                DEFAULT_JITTER);

        fprintf(fd, "\nEncoding parameters (must match sender):\n");
        fprintf(fd, "  -r <rate>   Sample rate (default %dHz)\n"
```

Figure 3.7: Rx (d)



```
\n",
                DEFAULT_JITTER);

        fprintf(fd, "\nEncoding parameters (must match sender):\n");
        fprintf(fd, "  -r <rate>    Sample rate (default %dHz)\n",
                DEFAULT_RATE);
        fprintf(fd, "  -c <n>       Number of channels (default %d)\n",
                DEFAULT_CHANNELS);

        fprintf(fd, "\nProgram parameters:\n");
        fprintf(fd, "  -v <n>       Verbosity level (default %d)\n",
                DEFAULT_VERBOSE);
        fprintf(fd, "  -D <file>    Run as a daemon, writing process ID t
the given file\n");
}

int main(int argc, char *argv[])
{
        int r, error;
        snd_pcm_t *snd;
        OpusDecoder *decoder;
        RtpSession *session;

        /* command-line options */
        const char *device = DEFAULT_DEVICE,
                *addr = DEFAULT_ADDR,
                *pid = NULL;
        unsigned int buffer = DEFAULT_BUFFER,
                rate = DEFAULT_RATE,
                jitter = DEFAULT_JITTER,
                channels = DEFAULT_CHANNELS,
                port = DEFAULT_PORT;

        fputs(COPYRIGHT "\n", stderr);

        for (;;) {
                int c;
```

Figure 3.7: Rx (e)



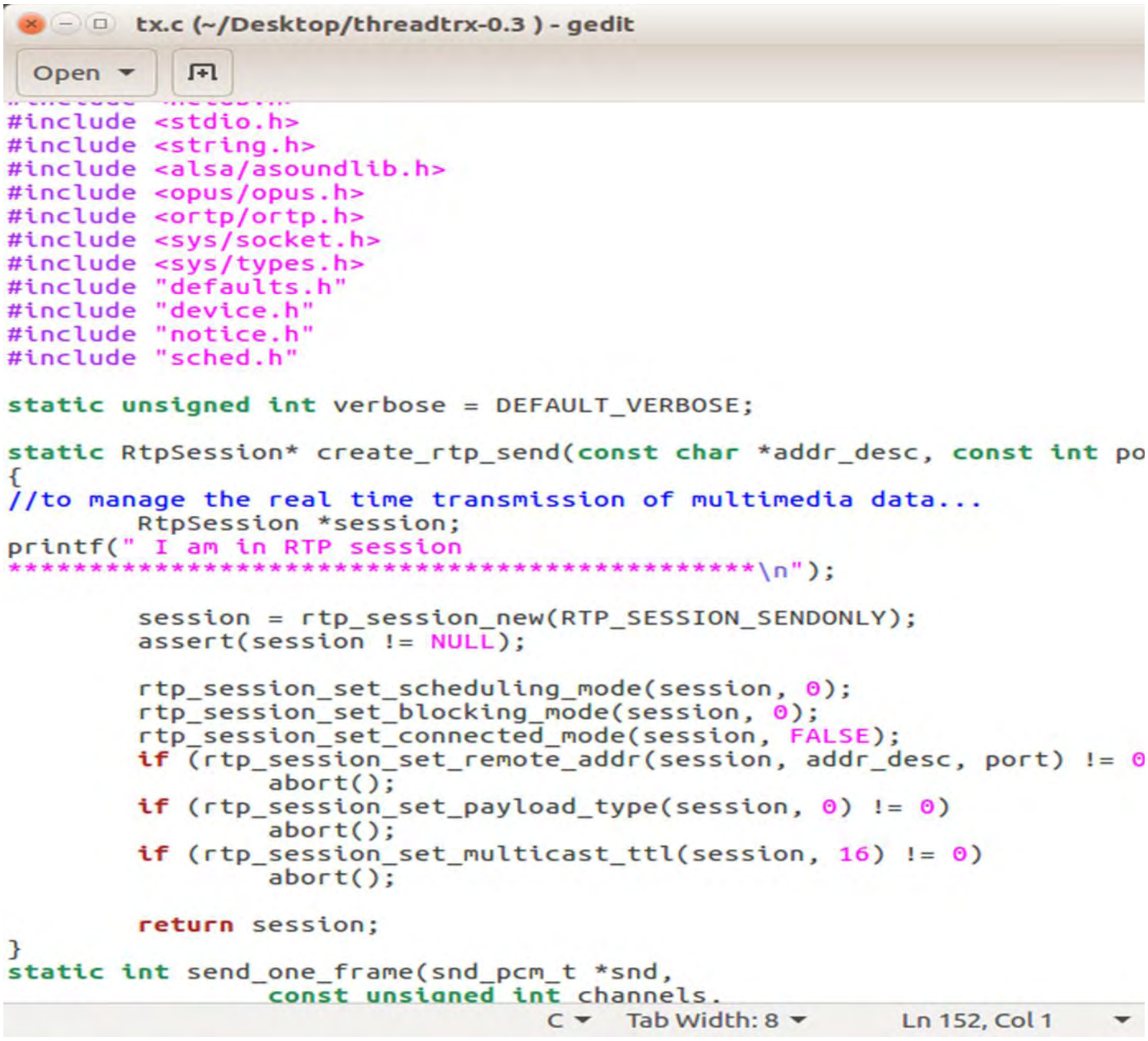
```
                fprintf(stderr, "opus_decoder_create: %s\n",
                        opus_strerror(error));
                return -1;
        }

        ortp_init();
        ortp_scheduler_init();
        session = create_rtp_recv(addr, port, jitter);
        assert(session != NULL);

        r = snd_pcm_open(&snd, device, SND_PCM_STREAM_PLAYBACK, 0);
        if (r < 0) {
                aerror("snd_pcm_open", r);
                return -1;
        }
        if (set_alsa_hw(snd, rate, channels, buffer * 1000) == -1)
                return -1;
        if (set_alsa_sw(snd) == -1)
                return -1;

        if (pid)
                go_daemon(pid);

        go_realtime();
        r = run_rx(session, decoder, snd, channels, rate);

        if (snd_pcm_close(snd) < 0)
                abort();

        rtp_session_destroy(session);
        ortp_exit();
        ortp_global_stats_display();

        opus_decoder_destroy(decoder);

        return r;
```

Figure 3.7: Rx (f)

3.3.2 THREADS

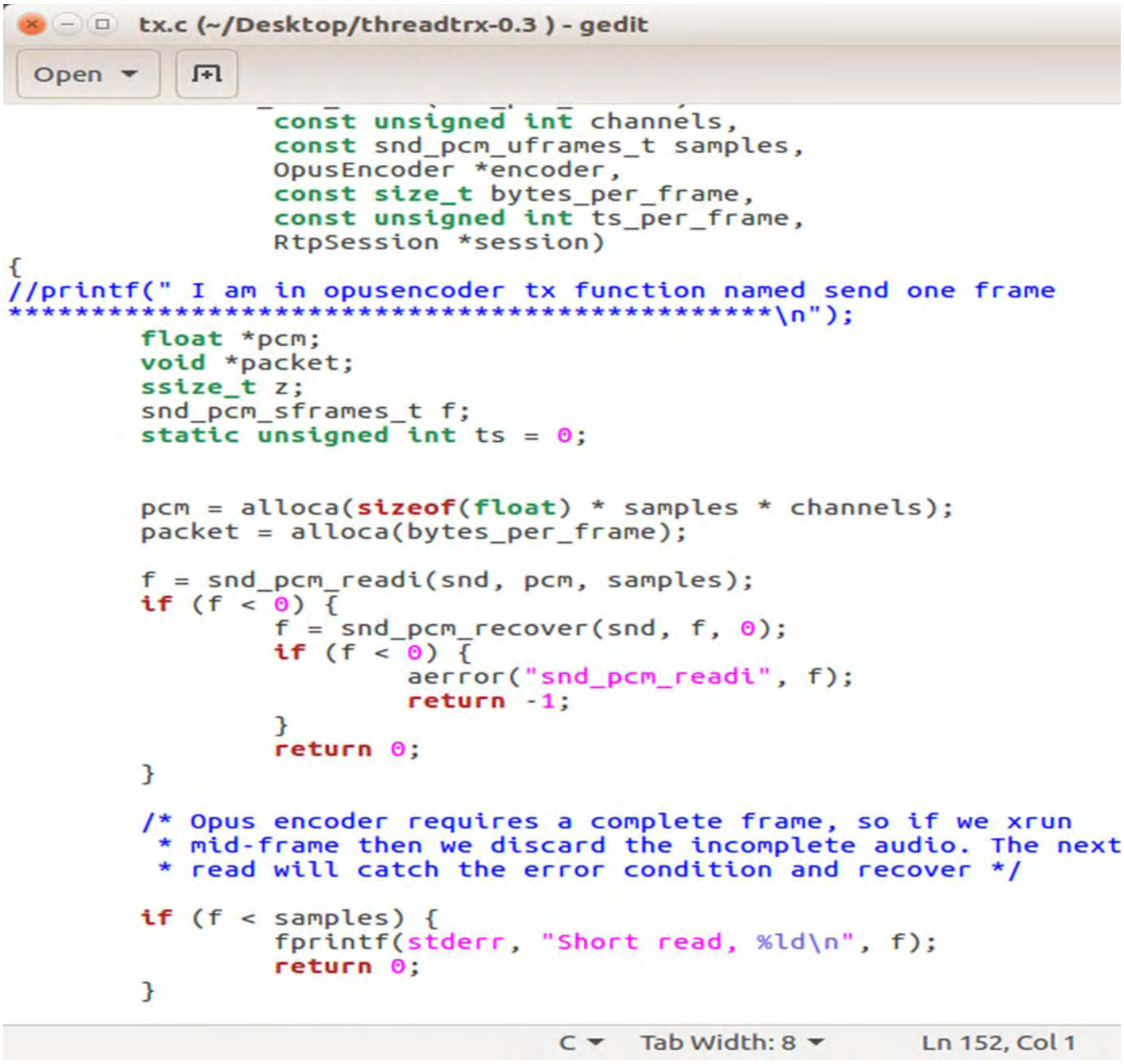
After doing half duplex, the next task is to applying the threads. A thread actually defines the path for execution of program. Each thread has a unique path for controlling the flow. Threads are light weighted process; one of the common examples to use thread is the implementation of concurrent programming done by operating system. Threads also used to save the wastage of CPU cycles and increasing the states of application. In simple words, threading is actually a parallel work which performs multiple tasks at the same time.

For applying the thread, it is necessary to install the ‘pthread’ library in Linux. It is necessary to link pthread library externally and use in Makefile. In half duplex threading is applied, the code is given in Figure 3.6 in half duplexing, two threads are used.



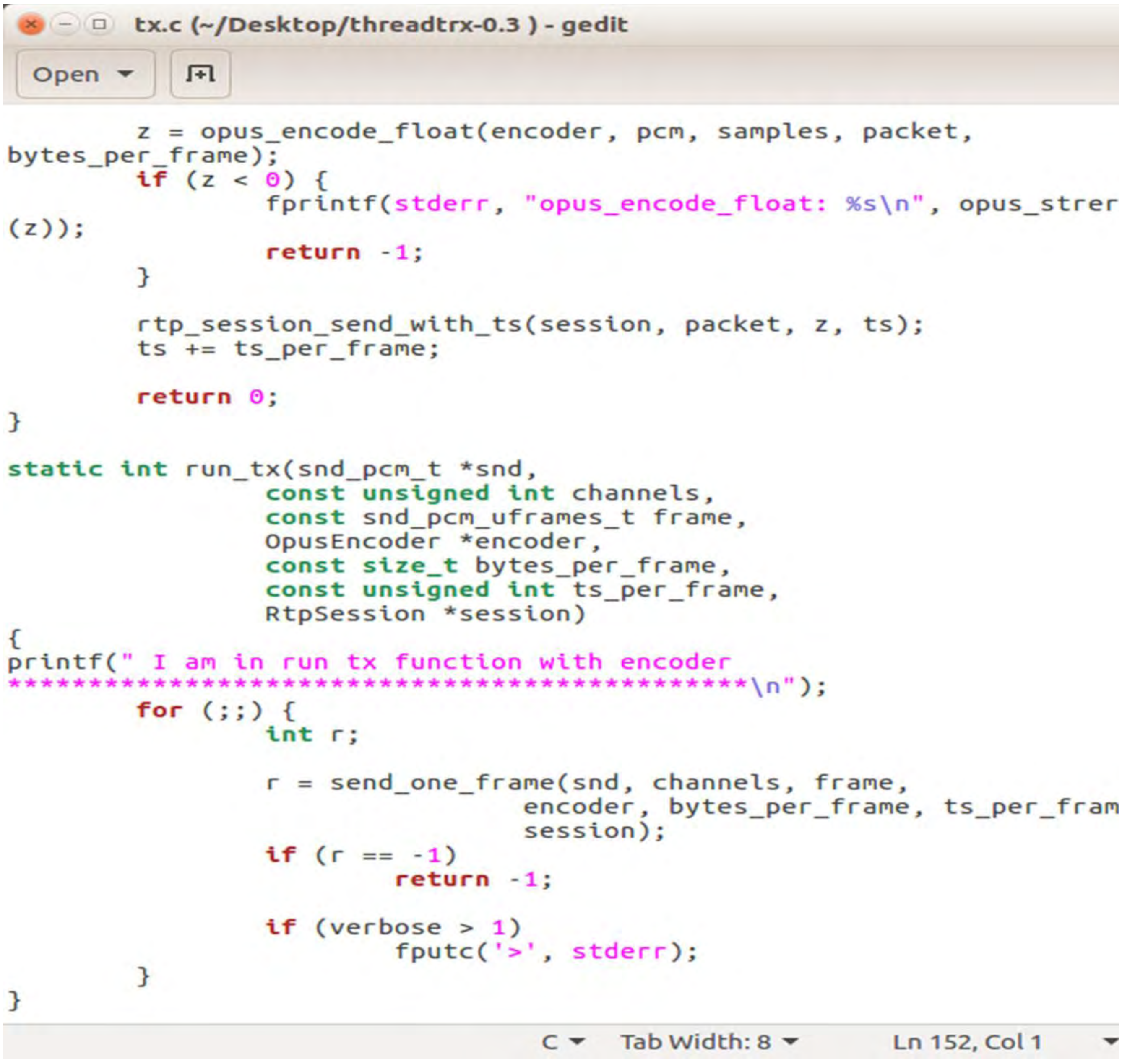
```
#include <stdio.h>
#include <string.h>
#include <alsa/asoundlib.h>
#include <opus/opus.h>
#include <ortp/ortp.h>
#include <sys/socket.h>
#include <sys/types.h>
#include "defaults.h"
#include "device.h"
#include "notice.h"
#include "sched.h"

static unsigned int verbose = DEFAULT_VERBOSE;

static RtpSession* create_rtp_send(const char *addr_desc, const int po
{
//to manage the real time transmission of multimedia data...
        RtpSession *session;
printf(" I am in RTP session
**********************************************\n");

        session = rtp_session_new(RTP_SESSION_SENDONLY);
        assert(session != NULL);

        rtp_session_set_scheduling_mode(session, 0);
        rtp_session_set_blocking_mode(session, 0);
        rtp_session_set_connected_mode(session, FALSE);
        if (rtp_session_set_remote_addr(session, addr_desc, port) != 0
                abort();
        if (rtp_session_set_payload_type(session, 0) != 0)
                abort();
        if (rtp_session_set_multicast_ttl(session, 16) != 0)
                abort();

        return session;
}
static int send_one_frame(snd_pcm_t *snd,
                const unsigned int channels,
```

Figure 3.8: Half duplex threading (a)



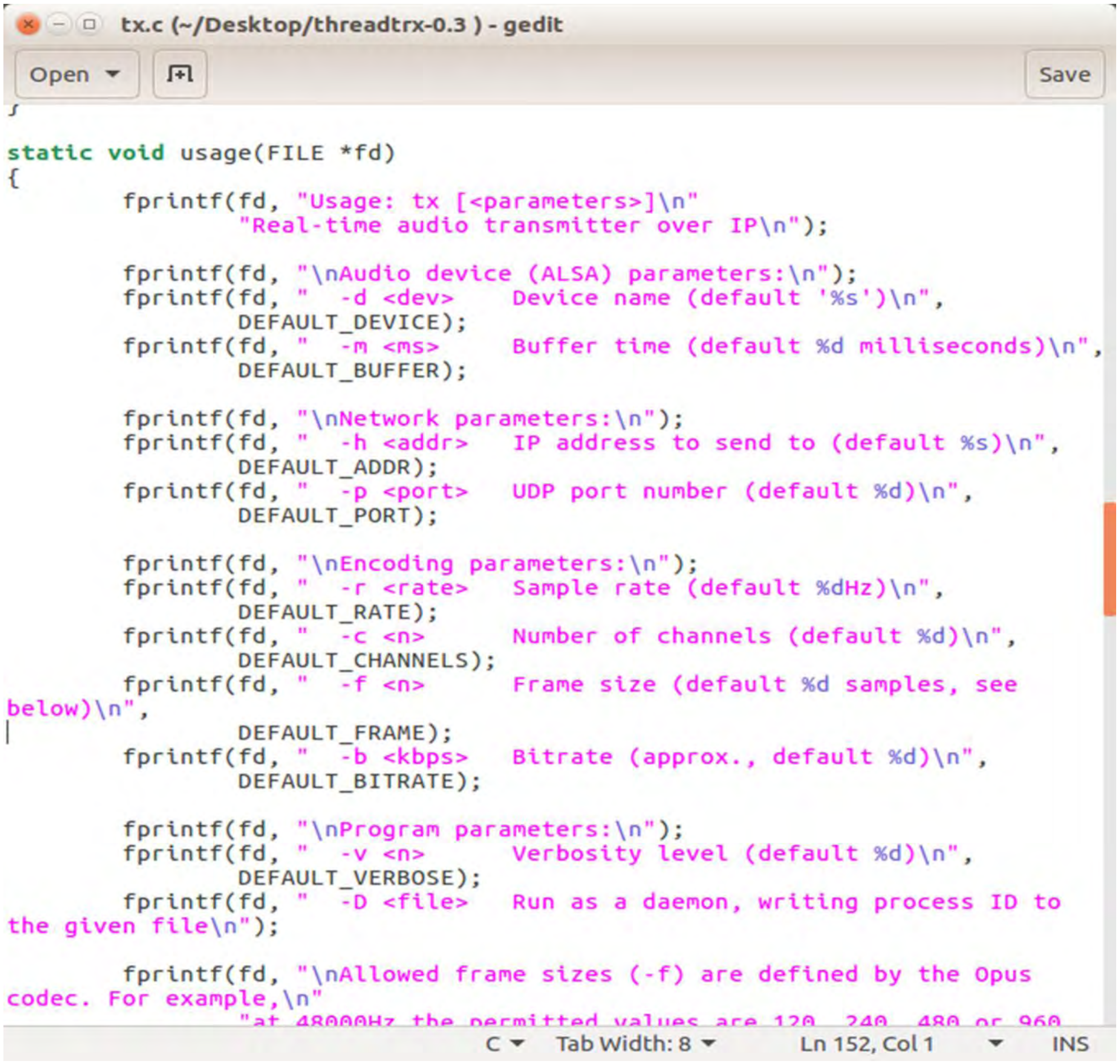
```
                    const unsigned int channels,
                    const snd_pcm_uframes_t samples,
                    OpusEncoder *encoder,
                    const size_t bytes_per_frame,
                    const unsigned int ts_per_frame,
                    RtpSession *session)
{
//printf(" I am in opusencoder tx function named send one frame
**********************************************\n");
        float *pcm;
        void *packet;
        ssize_t z;
        snd_pcm_sframes_t f;
        static unsigned int ts = 0;


        pcm = alloca(sizeof(float) * samples * channels);
        packet = alloca(bytes_per_frame);

        f = snd_pcm_readi(snd, pcm, samples);
        if (f < 0) {
                f = snd_pcm_recover(snd, f, 0);
                if (f < 0) {
                        aerror("snd_pcm_readi", f);
                        return -1;
                }
                return 0;
        }

        /* Opus encoder requires a complete frame, so if we xrun
         * mid-frame then we discard the incomplete audio. The next
         * read will catch the error condition and recover */

        if (f < samples) {
                fprintf(stderr, "Short read, %ld\n", f);
                return 0;
        }
```

Figure 3.8: Half duplex threading(b)



```
        z = opus_encode_float(encoder, pcm, samples, packet,
bytes_per_frame);
        if (z < 0) {
                fprintf(stderr, "opus_encode_float: %s\n", opus_strer
(z));
                return -1;
        }

        rtp_session_send_with_ts(session, packet, z, ts);
        ts += ts_per_frame;

        return 0;
}

static int run_tx(snd_pcm_t *snd,
                const unsigned int channels,
                const snd_pcm_uframes_t frame,
                OpusEncoder *encoder,
                const size_t bytes_per_frame,
                const unsigned int ts_per_frame,
                RtpSession *session)
{
printf(" I am in run tx function with encoder
**********************************************\n");
        for (;;) {
                int r;

                r = send_one_frame(snd, channels, frame,
                                encoder, bytes_per_frame, ts_per_fram
                                session);
                if (r == -1)
                        return -1;

                if (verbose > 1)
                        fputc('>', stderr);
        }
}
```

Figure 3.8: Half duplex threading(c)



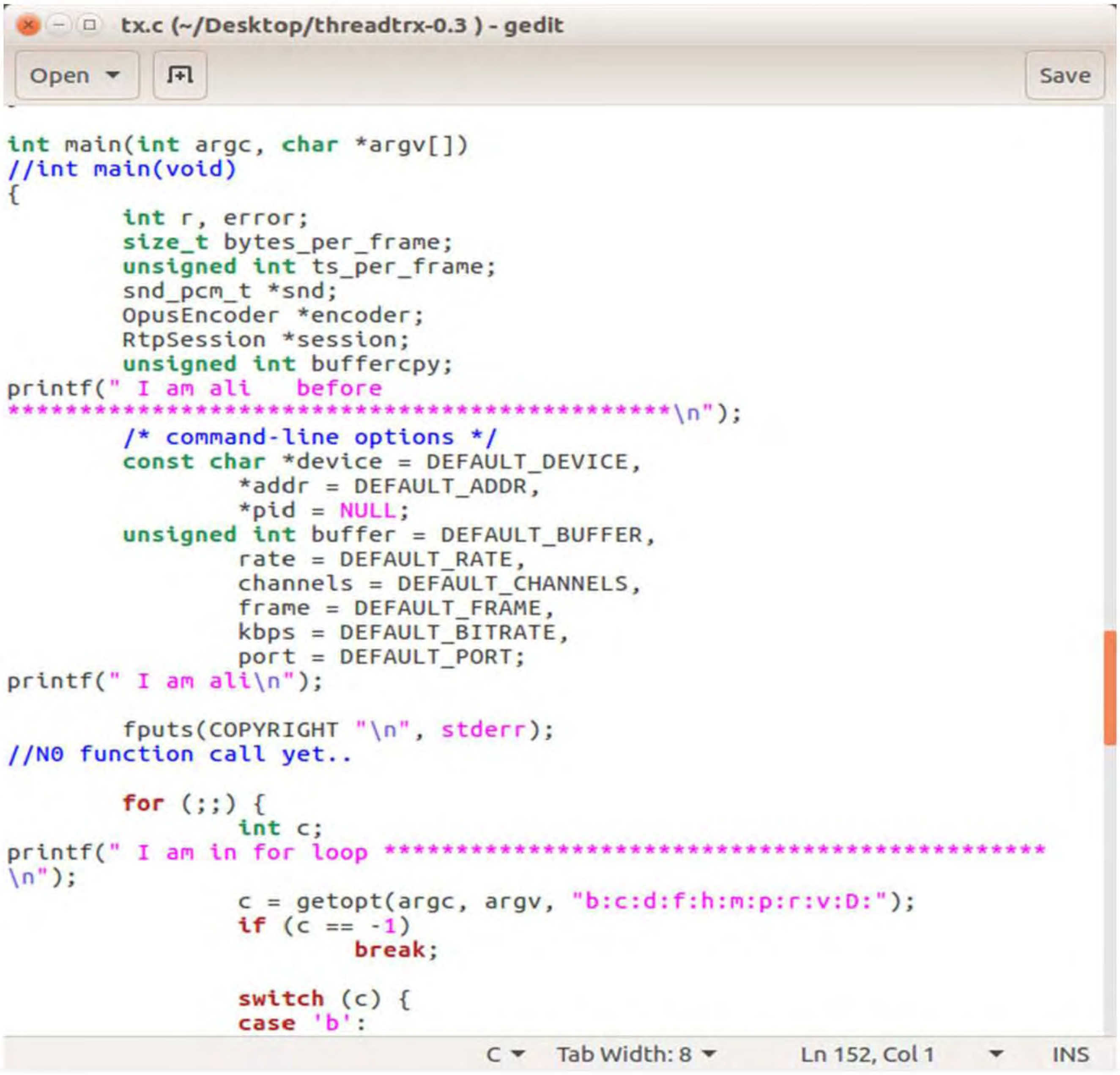
tx.c (~/Desktop/threadtrx-0.3) - gedit

Open Save

```
}

static void usage(FILE *fd)
{
        fprintf(fd, "Usage: tx [<parameters>]\n"
                "Real-time audio transmitter over IP\n");

        fprintf(fd, "\nAudio device (ALSA) parameters:\n");
        fprintf(fd, "  -d <dev>    Device name (default '%s')\n",
                DEFAULT_DEVICE);
        fprintf(fd, "  -m <ms>     Buffer time (default %d milliseconds)\n",
                DEFAULT_BUFFER);

        fprintf(fd, "\nNetwork parameters:\n");
        fprintf(fd, "  -h <addr>   IP address to send to (default %s)\n",
                DEFAULT_ADDR);
        fprintf(fd, "  -p <port>   UDP port number (default %d)\n",
                DEFAULT_PORT);

        fprintf(fd, "\nEncoding parameters:\n");
        fprintf(fd, "  -r <rate>   Sample rate (default %dHz)\n",
                DEFAULT_RATE);
        fprintf(fd, "  -c <n>      Number of channels (default %d)\n",
                DEFAULT_CHANNELS);
        fprintf(fd, "  -f <n>      Frame size (default %d samples, see
below)\n",
                DEFAULT_FRAME);
        fprintf(fd, "  -b <kbps>   Bitrate (approx., default %d)\n",
                DEFAULT_BITRATE);

        fprintf(fd, "\nProgram parameters:\n");
        fprintf(fd, "  -v <n>      Verbosity level (default %d)\n",
                DEFAULT_VERBOSE);
        fprintf(fd, "  -D <file>   Run as a daemon, writing process ID to
the given file\n");

        fprintf(fd, "\nAllowed frame sizes (-f) are defined by the Opus
codec. For example,\n"
                "at 48000Hz the permitted values are 120, 240, 480 or 960
```

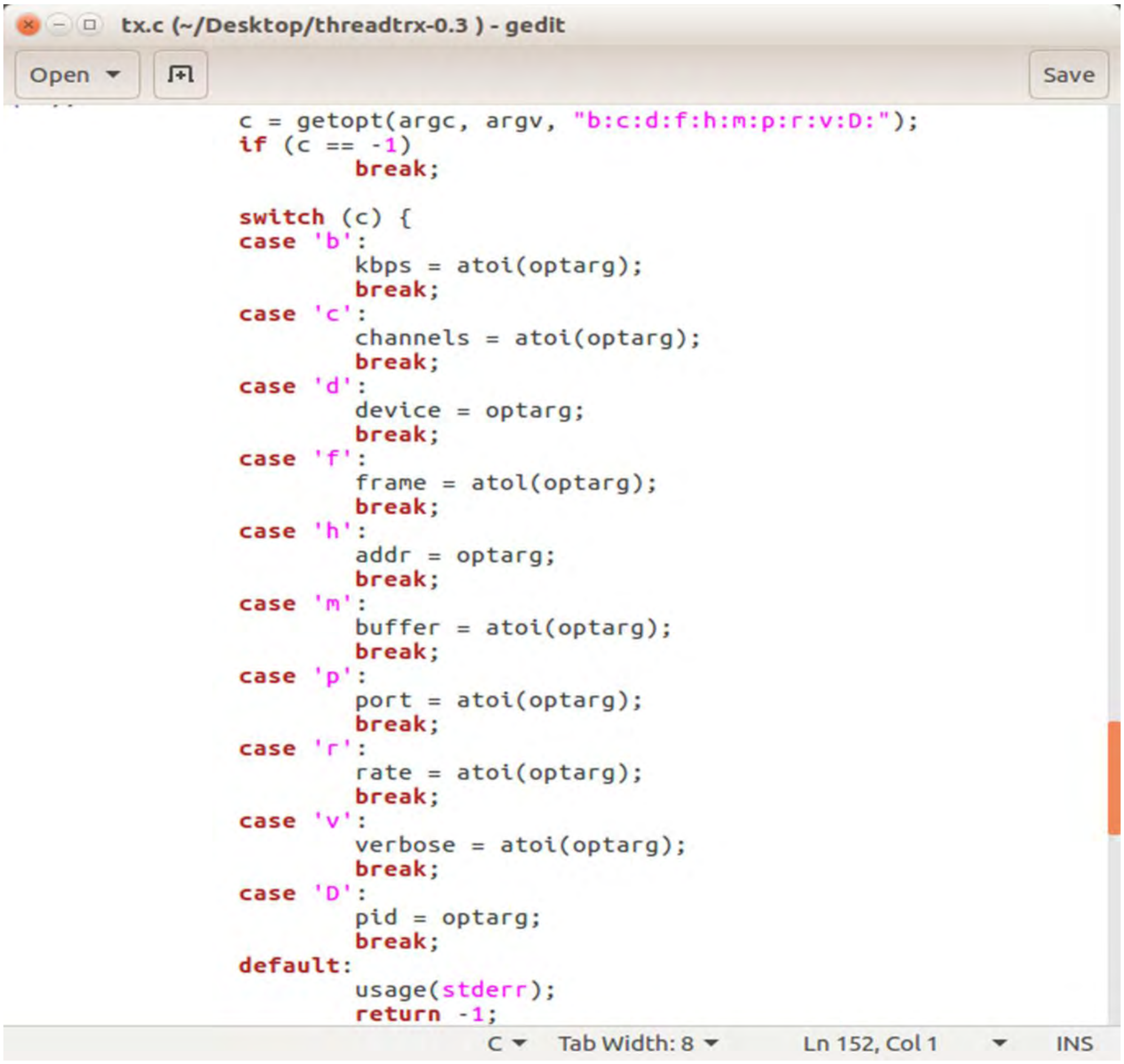
C Tab Width: 8 Ln 152, Col 1 INS

Figure 3.8: Half duplex threading(d)



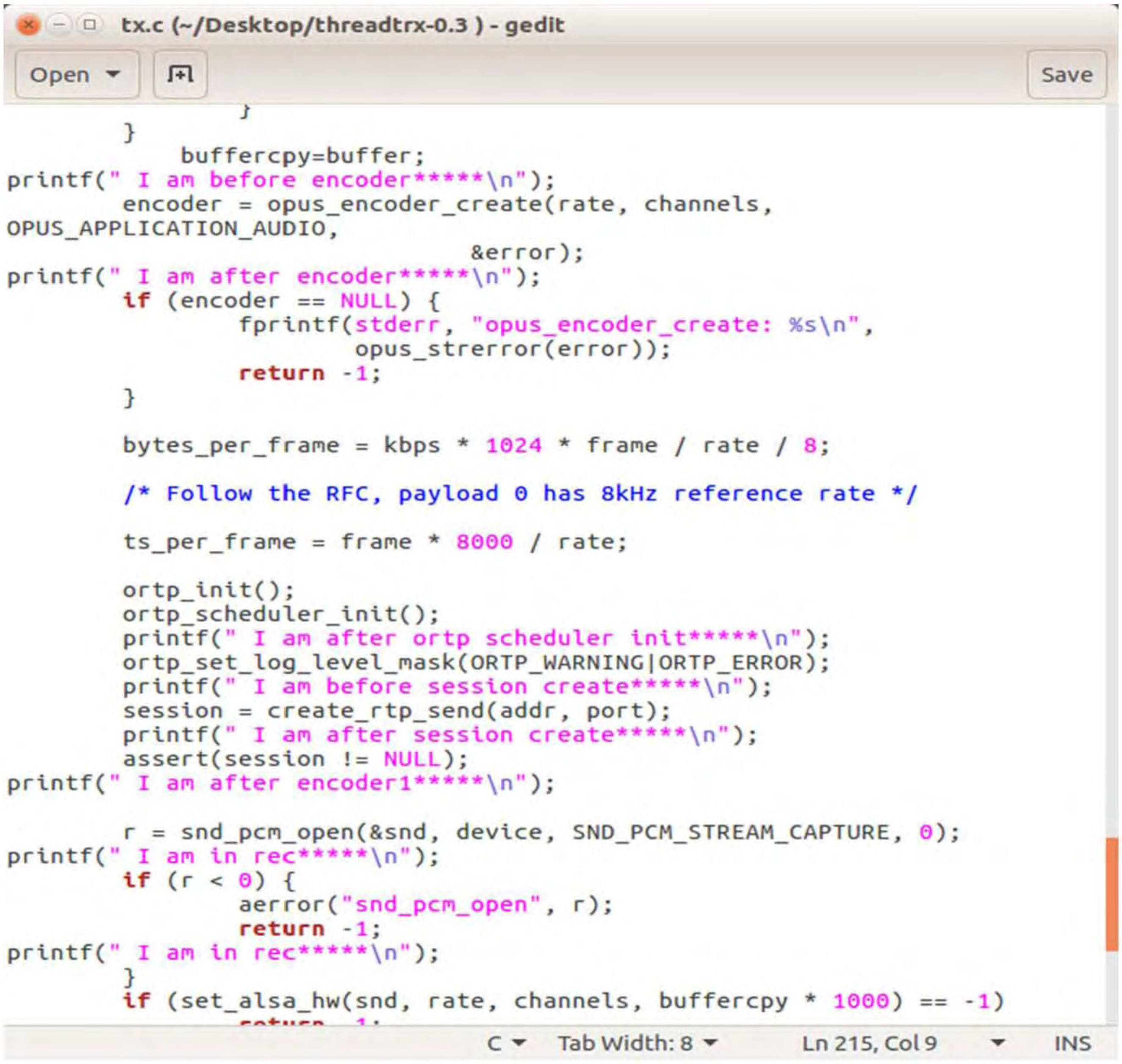
```
int main(int argc, char *argv[])
//int main(void)
{
        int r, error;
        size_t bytes_per_frame;
        unsigned int ts_per_frame;
        snd_pcm_t *snd;
        OpusEncoder *encoder;
        RtpSession *session;
        unsigned int buffercpy;
printf(" I am ali    before
*********************************************\n");
        /* command-line options */
        const char *device = DEFAULT_DEVICE,
                *addr = DEFAULT_ADDR,
                *pid = NULL;
        unsigned int buffer = DEFAULT_BUFFER,
                rate = DEFAULT_RATE,
                channels = DEFAULT_CHANNELS,
                frame = DEFAULT_FRAME,
                kbps = DEFAULT_BITRATE,
                port = DEFAULT_PORT;
printf(" I am ali\n");

        fputs(COPYRIGHT "\n", stderr);
//N0 function call yet..

        for (;;) {
                int c;
printf(" I am in for loop *********************************************
\n");
                c = getopt(argc, argv, "b:c:d:f:h:m:p:r:v:D:");
                if (c == -1)
                        break;

                switch (c) {
                case 'b':
```

Figure 3.8: Half duplex threading (e)



```
                c = getopt(argc, argv, "b:c:d:f:h:m:p:r:v:D:");
                if (c == -1)
                        break;

                switch (c) {
                case 'b':
                        kbps = atoi(optarg);
                        break;
                case 'c':
                        channels = atoi(optarg);
                        break;
                case 'd':
                        device = optarg;
                        break;
                case 'f':
                        frame = atol(optarg);
                        break;
                case 'h':
                        addr = optarg;
                        break;
                case 'm':
                        buffer = atoi(optarg);
                        break;
                case 'p':
                        port = atoi(optarg);
                        break;
                case 'r':
                        rate = atoi(optarg);
                        break;
                case 'v':
                        verbose = atoi(optarg);
                        break;
                case 'D':
                        pid = optarg;
                        break;
                default:
                        usage(stderr);
                        return -1;
```

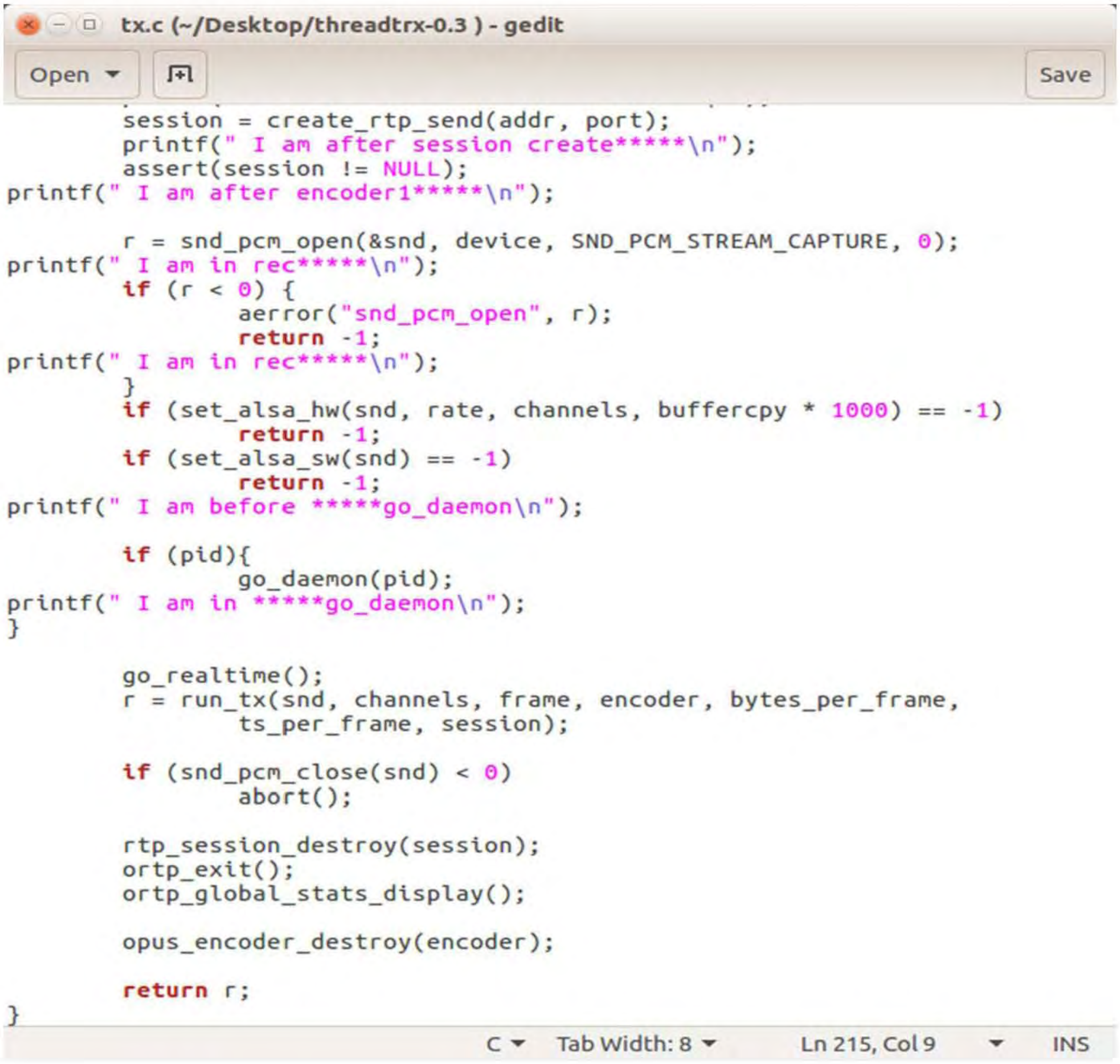
Figure 3.8: Half duplex threading (f)



tx.c (~/Desktop/threadtrx-0.3) - gedit

```
        }
            buffercpy=buffer;
printf(" I am before encoder*****\n");
        encoder = opus_encoder_create(rate, channels,
OPUS_APPLICATION_AUDIO,
                                    &error);
printf(" I am after encoder*****\n");
        if (encoder == NULL) {
                fprintf(stderr, "opus_encoder_create: %s\n",
                        opus_strerror(error));
                return -1;
        }

        bytes_per_frame = kbps * 1024 * frame / rate / 8;

        /* Follow the RFC, payload 0 has 8kHz reference rate */

        ts_per_frame = frame * 8000 / rate;

        ortp_init();
        ortp_scheduler_init();
        printf(" I am after ortp scheduler init*****\n");
        ortp_set_log_level_mask(ORTP_WARNING|ORTP_ERROR);
        printf(" I am before session create*****\n");
        session = create_rtp_send(addr, port);
        printf(" I am after session create*****\n");
        assert(session != NULL);
printf(" I am after encoder1*****\n");

        r = snd_pcm_open(&snd, device, SND_PCM_STREAM_CAPTURE, 0);
printf(" I am in rec*****\n");
        if (r < 0) {
                aerror("snd_pcm_open", r);
                return -1;
printf(" I am in rec*****\n");
        }
        if (set_alsa_hw(snd, rate, channels, buffercpy * 1000) == -1)
```

Figure 3.8: Half duplex threading (g)



```
        session = create_rtp_send(addr, port);
        printf(" I am after session create*****\n");
        assert(session != NULL);
printf(" I am after encoder1*****\n");

        r = snd_pcm_open(&snd, device, SND_PCM_STREAM_CAPTURE, 0);
printf(" I am in rec*****\n");
        if (r < 0) {
                aerror("snd_pcm_open", r);
                return -1;
printf(" I am in rec*****\n");
        }
        if (set_alsa_hw(snd, rate, channels, buffercpy * 1000) == -1)
                return -1;
        if (set_alsa_sw(snd) == -1)
                return -1;
printf(" I am before *****go_daemon\n");

        if (pid){
                go_daemon(pid);
printf(" I am in *****go_daemon\n");
}

        go_realtime();
        r = run_tx(snd, channels, frame, encoder, bytes_per_frame,
                ts_per_frame, session);

        if (snd_pcm_close(snd) < 0)
                abort();

        rtp_session_destroy(session);
        ortp_exit();
        ortp_global_stats_display();

        opus_encoder_destroy(encoder);

        return r;
}
```

Figure 3.8: Half duplex threading (h)

3.3.3 FULL DUPLEX

In full duplex, both transmitter and receiver can send and receive the data simultaneously. Full duplexing is bidirectional data is transmitted, like two way traffic on road. The whole process of full duplex is same as explain in above in half duplex. In half duplex, data is transmitting only in one side and the other end only receive the data, but in full duplex data transmitted simultaneously. The main process of full duplexing is given in Figure 3.7.

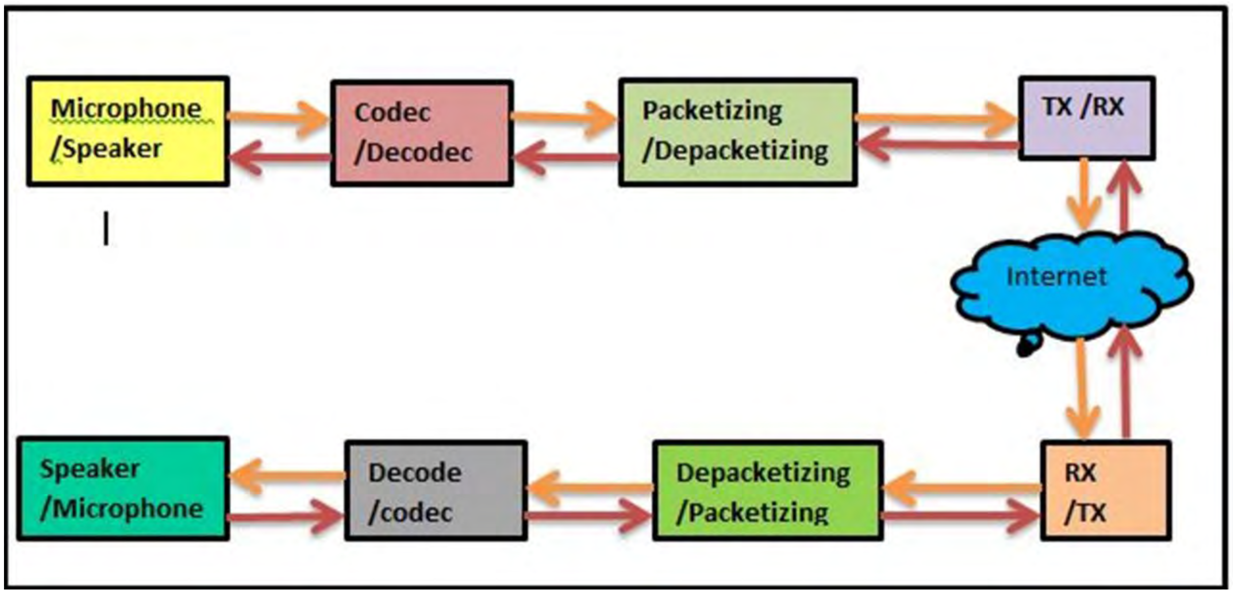
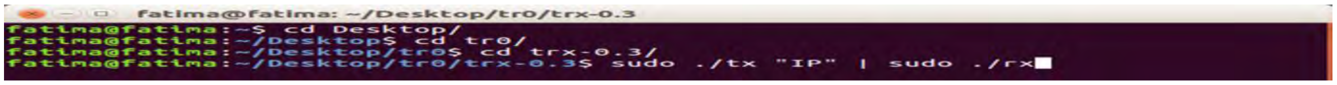


Figure 3.7: Full duplex

The difference in half and full duplex is the only that in full duplex both devices act as sender and a receiver at a time. The code for the full duplex is same as for the half duplex. Same code of half duplexing is used for full duplexing, to run the code for full duplex the following command is used which is given in Figure 3.8.

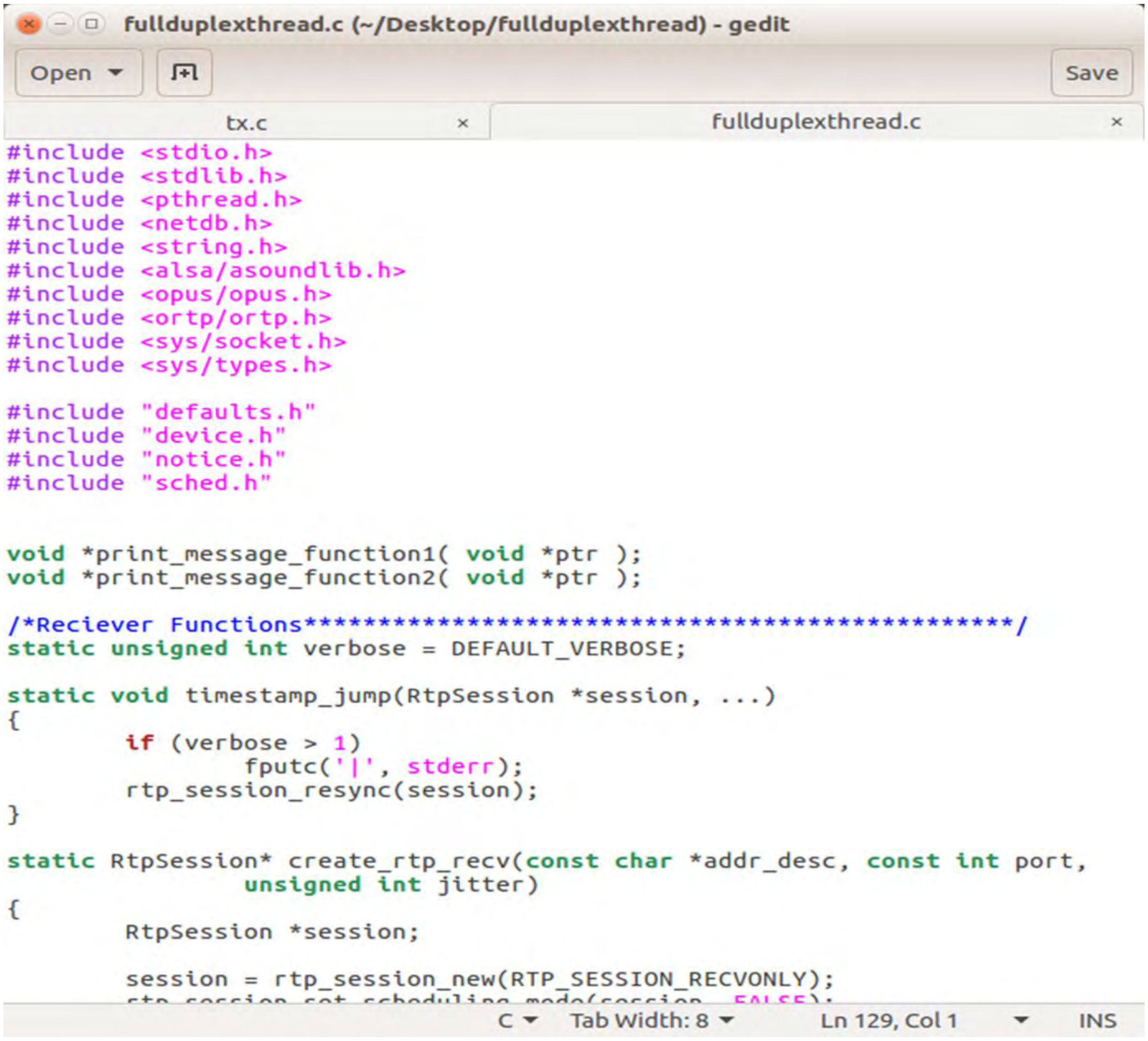


```
fatima@fatima: ~/Desktop/tr0/trx-0.3
fatima@fatima:~$ cd Desktop/
fatima@fatima:~/Desktop$ cd tr0/
fatima@fatima:~/Desktop/tr0$ cd trx-0.3/
fatima@fatima:~/Desktop/tr0/trx-0.3$ sudo ./tx "IP" | sudo ./rx
```

Figure 3.8: command of full duplex

3.3.4 THREADS

In order to optimize the usage of CPU, C language containing multi-threading features is used, which authorizes simultaneous programming of two or more parts of program. Therefore thread a part of program is a process within a process. In full duplexing, threads are applied the code of applying thread is given in Figure 3.8. In this two threads are used, One thread is used for transmission and the other one is used for receiving.



fullduplexthread.c (~/Desktop/fullduplexthread) - gedit

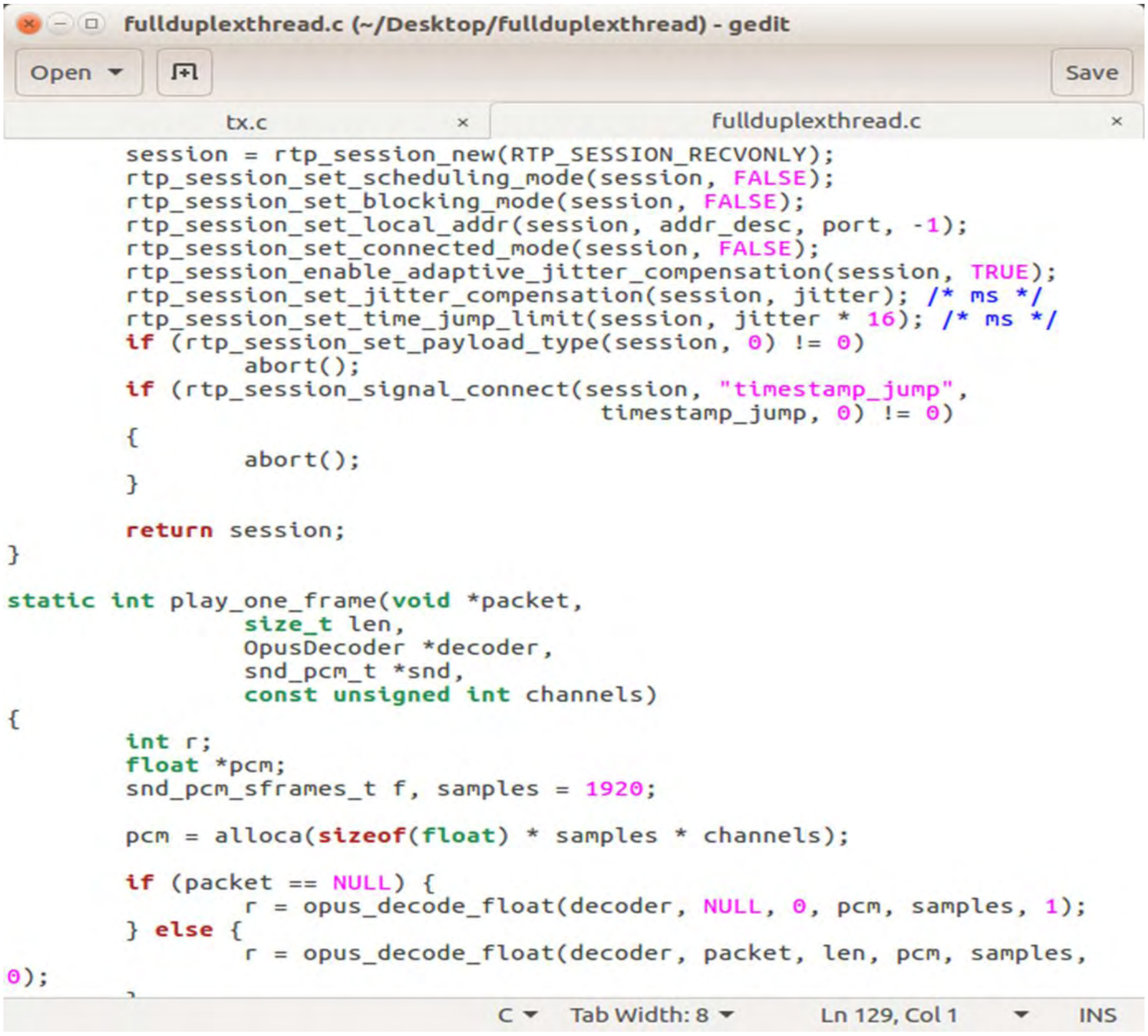
Open | Save

tx.c | fullduplexthread.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <netdb.h>
#include <string.h>
#include <alsa/asoundlib.h>
#include <opus/opus.h>
#include <ortp/ortp.h>
#include <sys/socket.h>
#include <sys/types.h>

#include "defaults.h"
#include "device.h"
#include "notice.h"
#include "sched.h"


void *print_message_function1( void *ptr );
void *print_message_function2( void *ptr );

/*Reciever Functions************************************************/
static unsigned int verbose = DEFAULT_VERBOSE;

static void timestamp_jump(RtpSession *session, ...)
{
        if (verbose > 1)
                fputc('|', stderr);
        rtp_session_resync(session);
}

static RtpSession* create_rtp_recv(const char *addr_desc, const int port,
                unsigned int jitter)
{
        RtpSession *session;

        session = rtp_session_new(RTP_SESSION_RECVONLY);
```

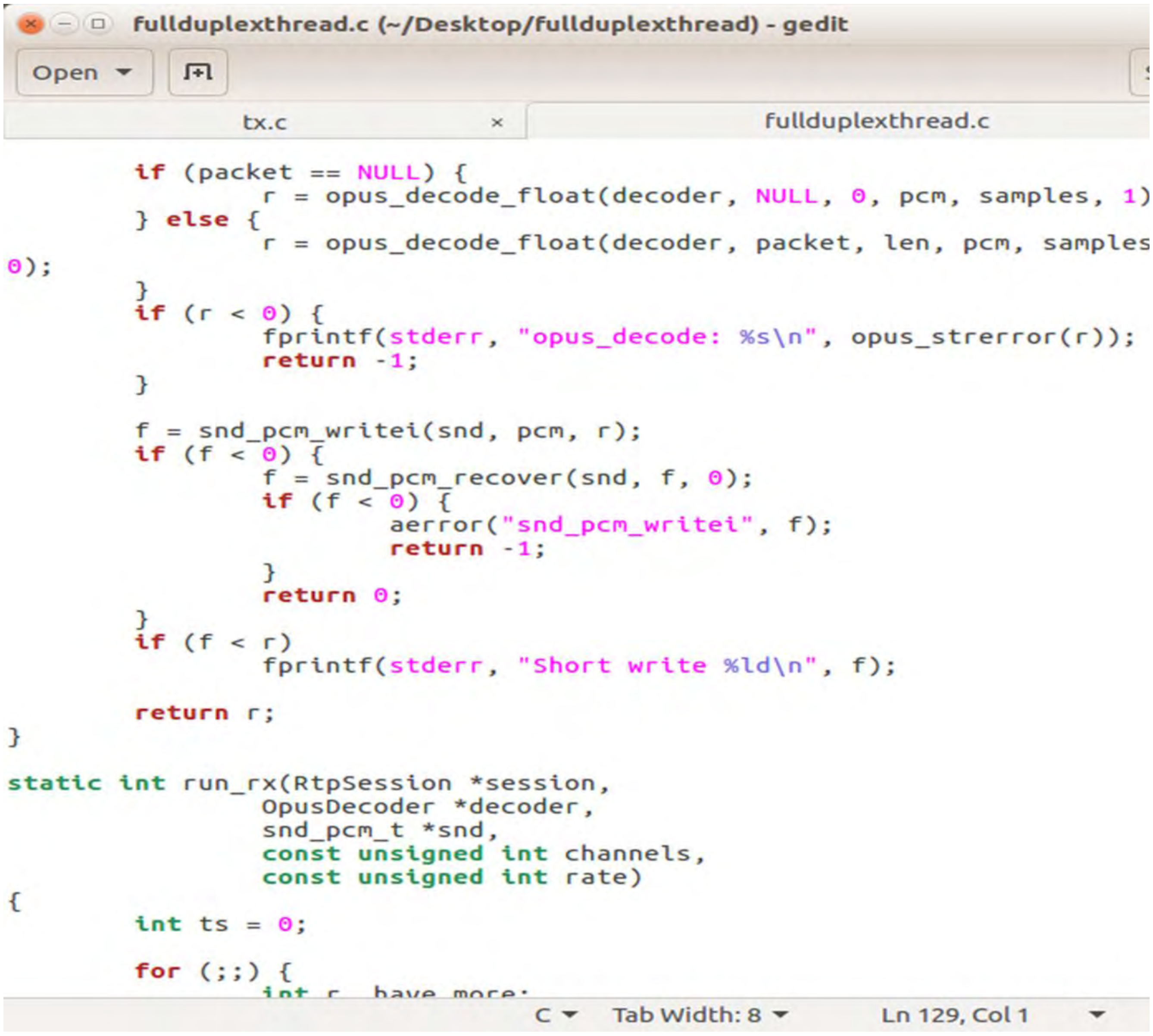
C ▾ Tab Width: 8 ▾ Ln 129, Col 1 ▾ INS

Figure 3.9: Full duplex threading(a)



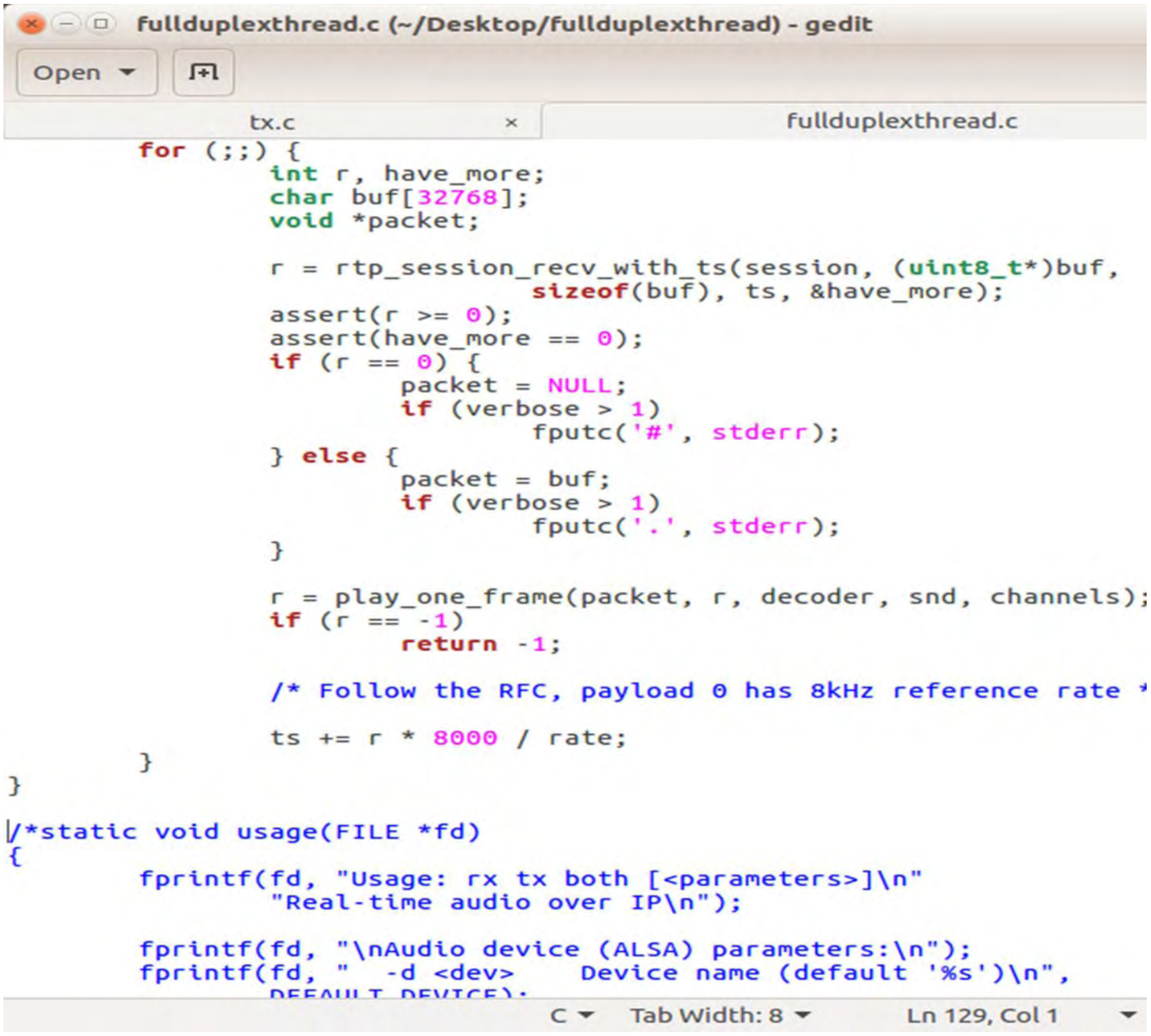
```
        session = rtp_session_new(RTP_SESSION_RECVONLY);
        rtp_session_set_scheduling_mode(session, FALSE);
        rtp_session_set_blocking_mode(session, FALSE);
        rtp_session_set_local_addr(session, addr_desc, port, -1);
        rtp_session_set_connected_mode(session, FALSE);
        rtp_session_enable_adaptive_jitter_compensation(session, TRUE);
        rtp_session_set_jitter_compensation(session, jitter); /* ms */
        rtp_session_set_time_jump_limit(session, jitter * 16); /* ms */
        if (rtp_session_set_payload_type(session, 0) != 0)
                abort();
        if (rtp_session_signal_connect(session, "timestamp_jump",
                                        timestamp_jump, 0) != 0)
        {
                abort();
        }

        return session;
}

static int play_one_frame(void *packet,
                size_t len,
                OpusDecoder *decoder,
                snd_pcm_t *snd,
                const unsigned int channels)
{
        int r;
        float *pcm;
        snd_pcm_sframes_t f, samples = 1920;

        pcm = alloca(sizeof(float) * samples * channels);

        if (packet == NULL) {
                r = opus_decode_float(decoder, NULL, 0, pcm, samples, 1);
        } else {
                r = opus_decode_float(decoder, packet, len, pcm, samples,
0);
```

Figure 3.9: Full duplex threading (b)



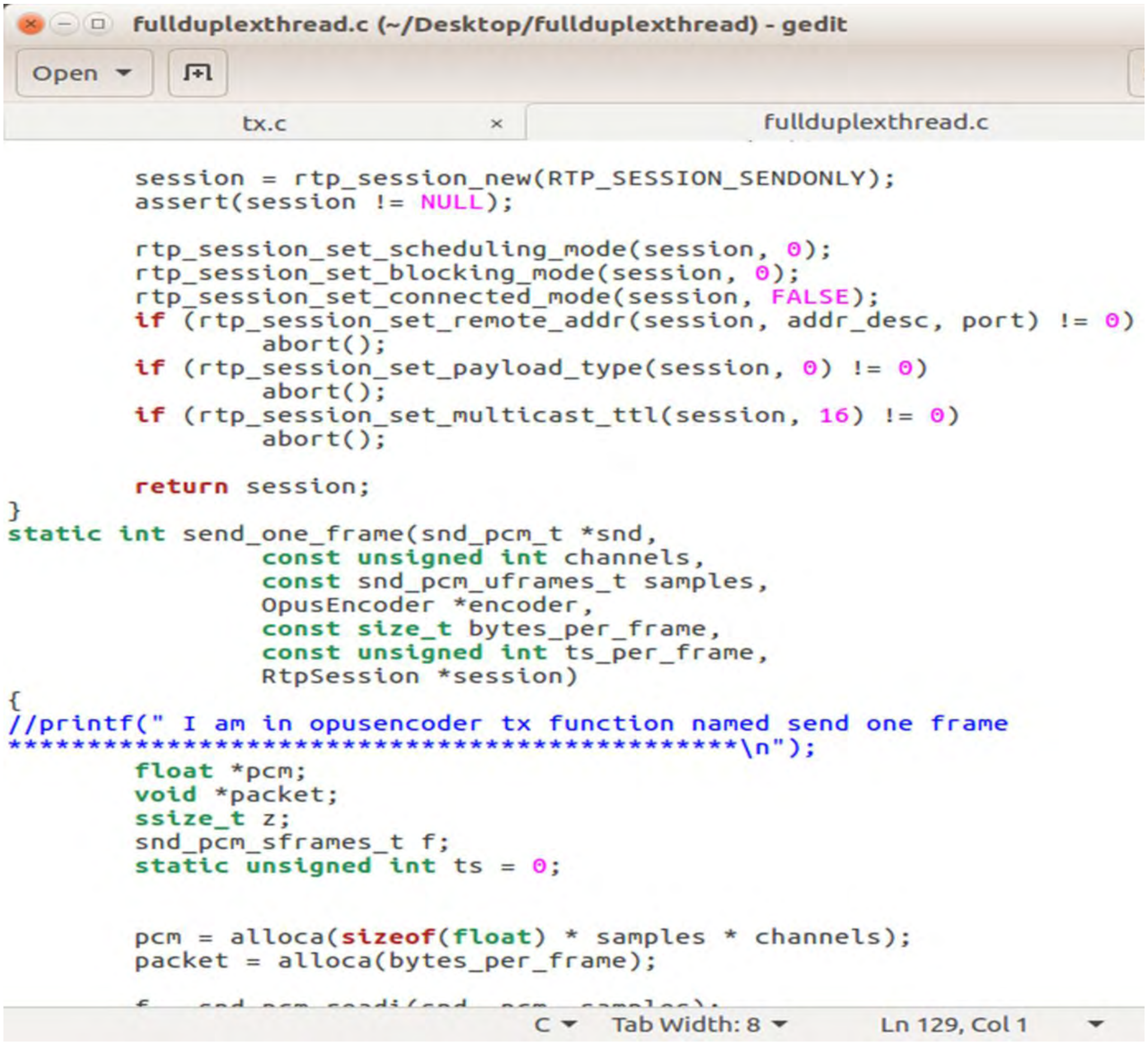
```
	if (packet == NULL) {
		r = opus_decode_float(decoder, NULL, 0, pcm, samples, 1)
	} else {
		r = opus_decode_float(decoder, packet, len, pcm, samples
0);
	}
	if (r < 0) {
		fprintf(stderr, "opus_decode: %s\n", opus_strerror(r));
		return -1;
	}

	f = snd_pcm_writei(snd, pcm, r);
	if (f < 0) {
		f = snd_pcm_recover(snd, f, 0);
		if (f < 0) {
			aerror("snd_pcm_writei", f);
			return -1;
		}
		return 0;
	}
	if (f < r)
		fprintf(stderr, "Short write %ld\n", f);

	return r;
}

static int run_rx(RtpSession *session,
		OpusDecoder *decoder,
		snd_pcm_t *snd,
		const unsigned int channels,
		const unsigned int rate)
{
	int ts = 0;

	for (;;) {
		int r, have_more;
```

Figure 3.9: Full duplex threading(c)



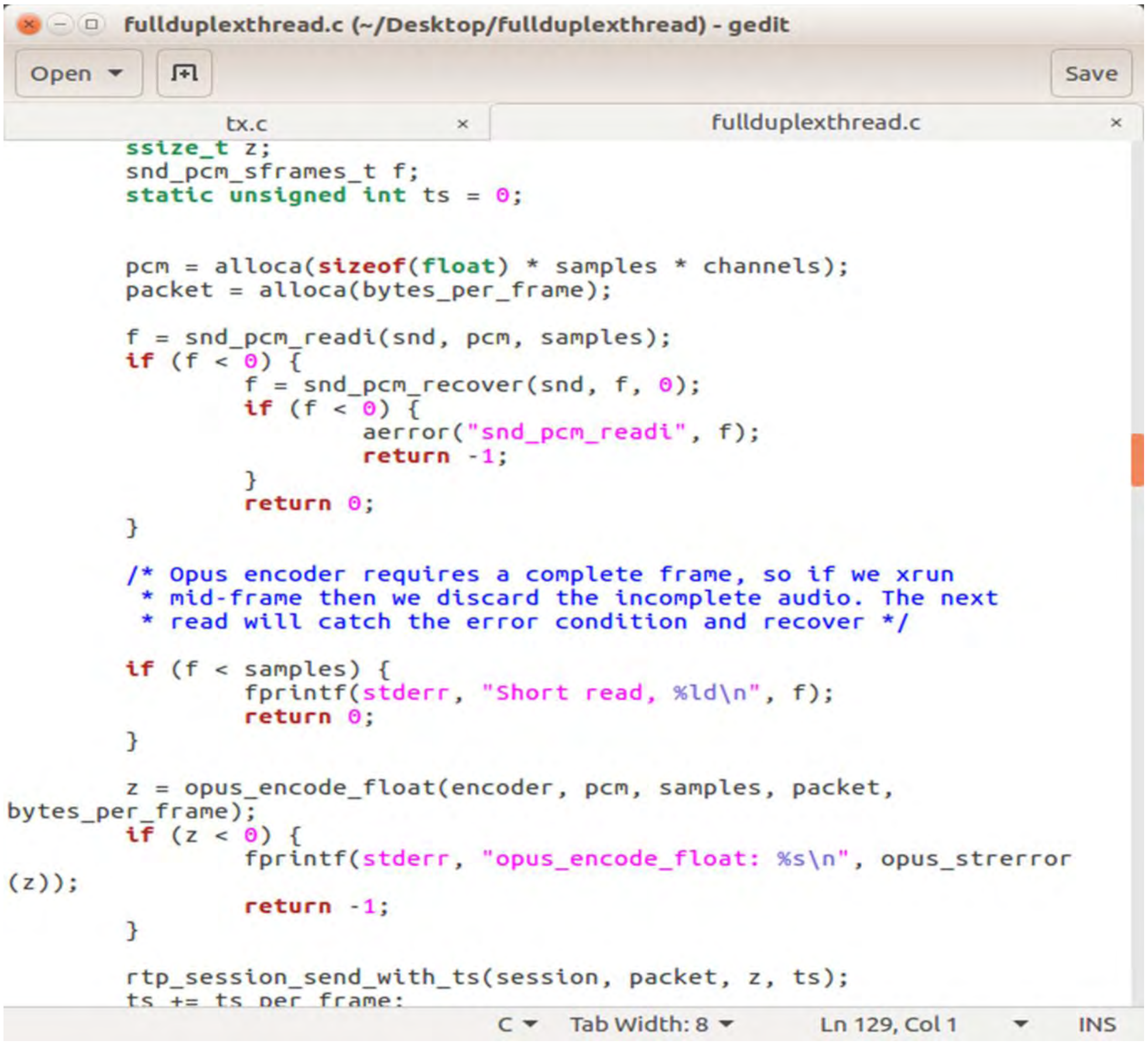
```
        for (;;) {
                int r, have_more;
                char buf[32768];
                void *packet;

                r = rtp_session_recv_with_ts(session, (uint8_t*)buf,
                                sizeof(buf), ts, &have_more);
                assert(r >= 0);
                assert(have_more == 0);
                if (r == 0) {
                        packet = NULL;
                        if (verbose > 1)
                                fputc('#', stderr);
                } else {
                        packet = buf;
                        if (verbose > 1)
                                fputc('.', stderr);
                }

                r = play_one_frame(packet, r, decoder, snd, channels);
                if (r == -1)
                        return -1;

                /* Follow the RFC, payload 0 has 8kHz reference rate *

                ts += r * 8000 / rate;
        }
}

/*static void usage(FILE *fd)
{
        fprintf(fd, "Usage: rx tx both [<parameters>]\n"
                "Real-time audio over IP\n");

        fprintf(fd, "\nAudio device (ALSA) parameters:\n");
        fprintf(fd, "  -d <dev>    Device name (default '%s')\n",
                DEFAULT_DEVICE);
```

Figure 3.9: Full duplex threading (d)



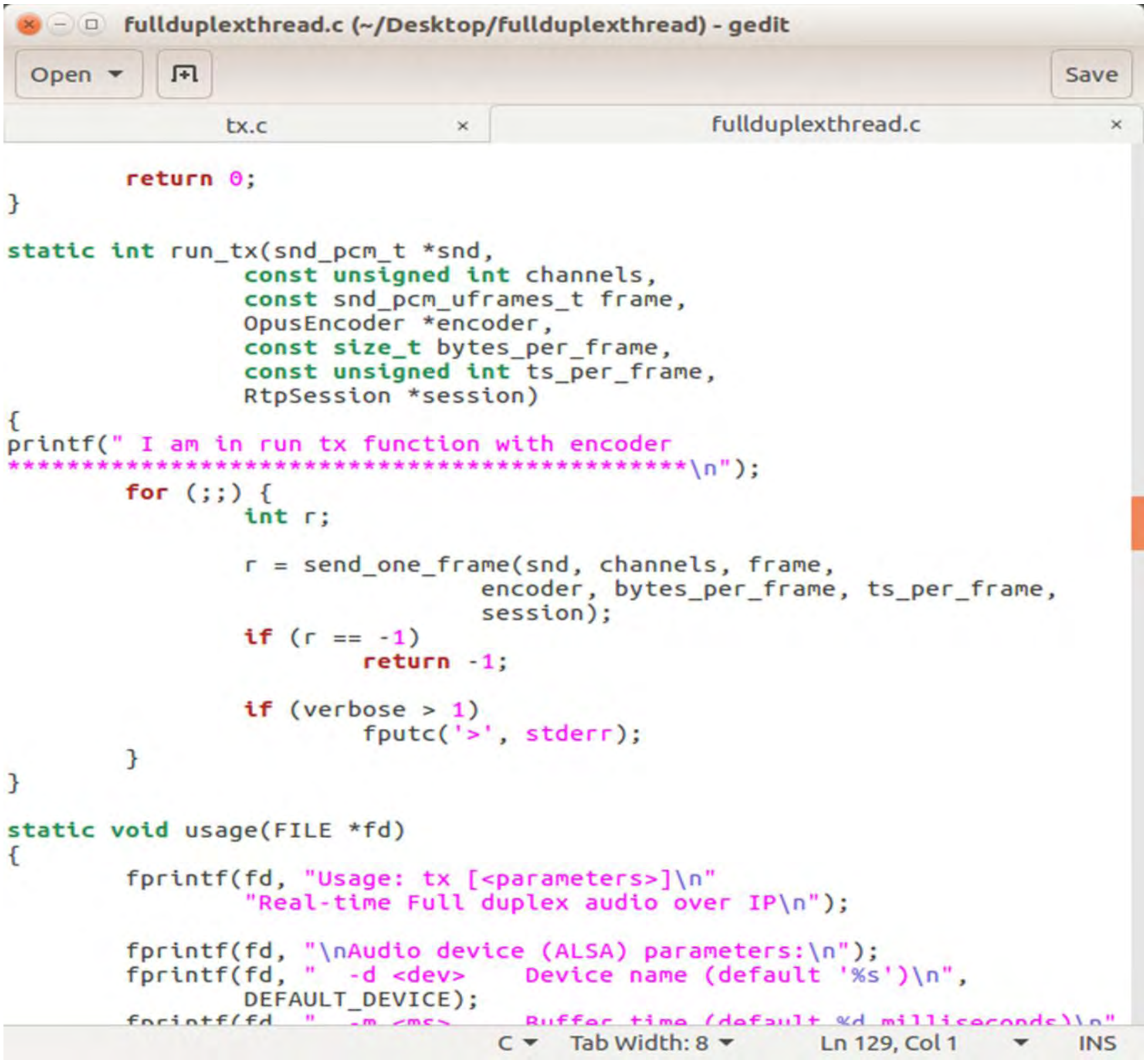
```
        session = rtp_session_new(RTP_SESSION_SENDONLY);
        assert(session != NULL);

        rtp_session_set_scheduling_mode(session, 0);
        rtp_session_set_blocking_mode(session, 0);
        rtp_session_set_connected_mode(session, FALSE);
        if (rtp_session_set_remote_addr(session, addr_desc, port) != 0)
                abort();
        if (rtp_session_set_payload_type(session, 0) != 0)
                abort();
        if (rtp_session_set_multicast_ttl(session, 16) != 0)
                abort();

        return session;
}
static int send_one_frame(snd_pcm_t *snd,
                const unsigned int channels,
                const snd_pcm_uframes_t samples,
                OpusEncoder *encoder,
                const size_t bytes_per_frame,
                const unsigned int ts_per_frame,
                RtpSession *session)
{
//printf(" I am in opusencoder tx function named send one frame
**********************************************\n");
        float *pcm;
        void *packet;
        ssize_t z;
        snd_pcm_sframes_t f;
        static unsigned int ts = 0;


        pcm = alloca(sizeof(float) * samples * channels);
        packet = alloca(bytes_per_frame);
```

Figure 3.9: Full duplex threading (e)



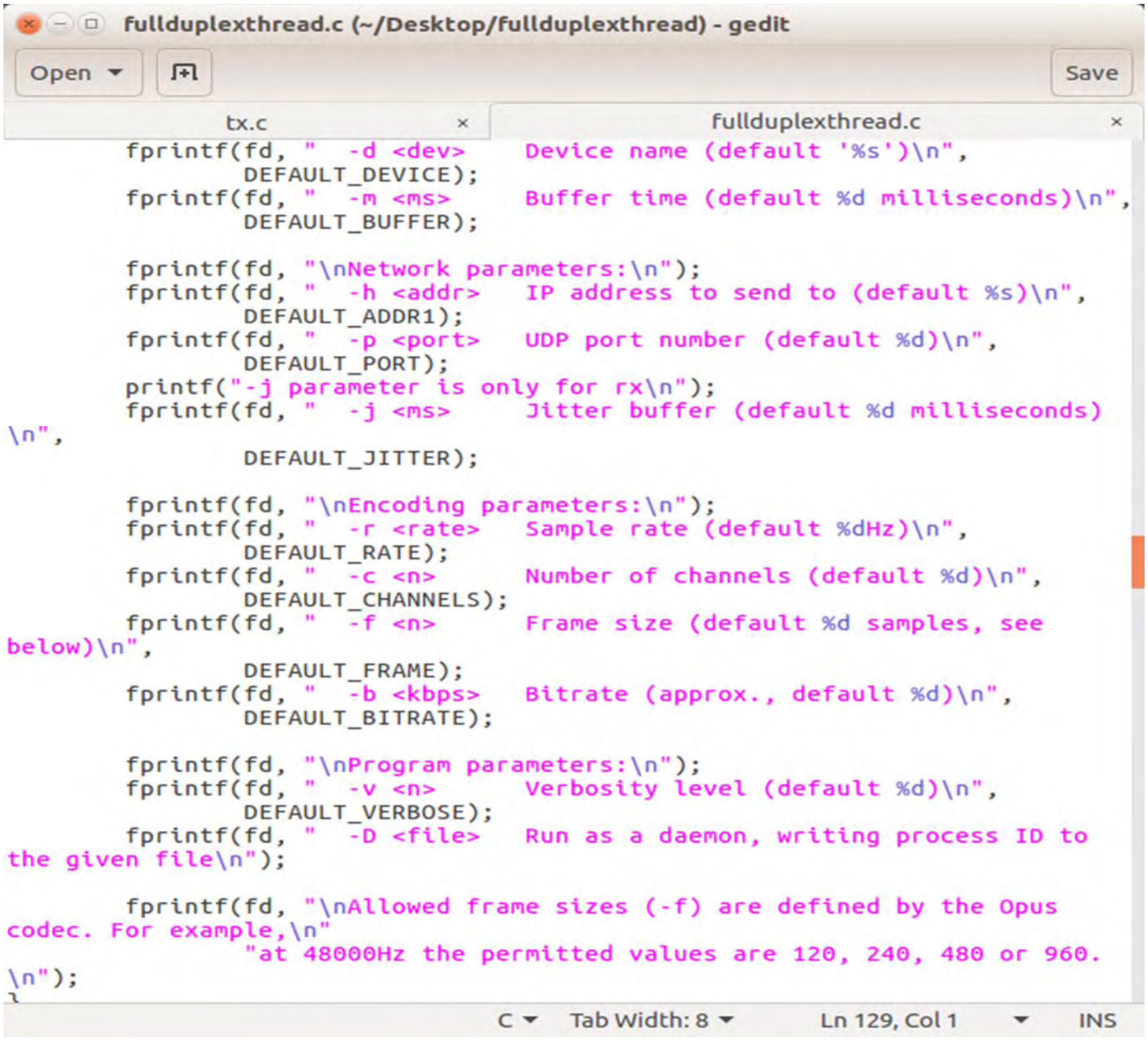
```c
        ssize_t z;
        snd_pcm_sframes_t f;
        static unsigned int ts = 0;


        pcm = alloca(sizeof(float) * samples * channels);
        packet = alloca(bytes_per_frame);

        f = snd_pcm_readi(snd, pcm, samples);
        if (f < 0) {
                f = snd_pcm_recover(snd, f, 0);
                if (f < 0) {
                        aerror("snd_pcm_readi", f);
                        return -1;
                }
                return 0;
        }

        /* Opus encoder requires a complete frame, so if we xrun
         * mid-frame then we discard the incomplete audio. The next
         * read will catch the error condition and recover */

        if (f < samples) {
                fprintf(stderr, "Short read, %ld\n", f);
                return 0;
        }

        z = opus_encode_float(encoder, pcm, samples, packet,
bytes_per_frame);
        if (z < 0) {
                fprintf(stderr, "opus_encode_float: %s\n", opus_strerror
(z));
                return -1;
        }

        rtp_session_send_with_ts(session, packet, z, ts);
        ts += ts_per_frame;
```

Figure 3.9: Full duplex threading (f)



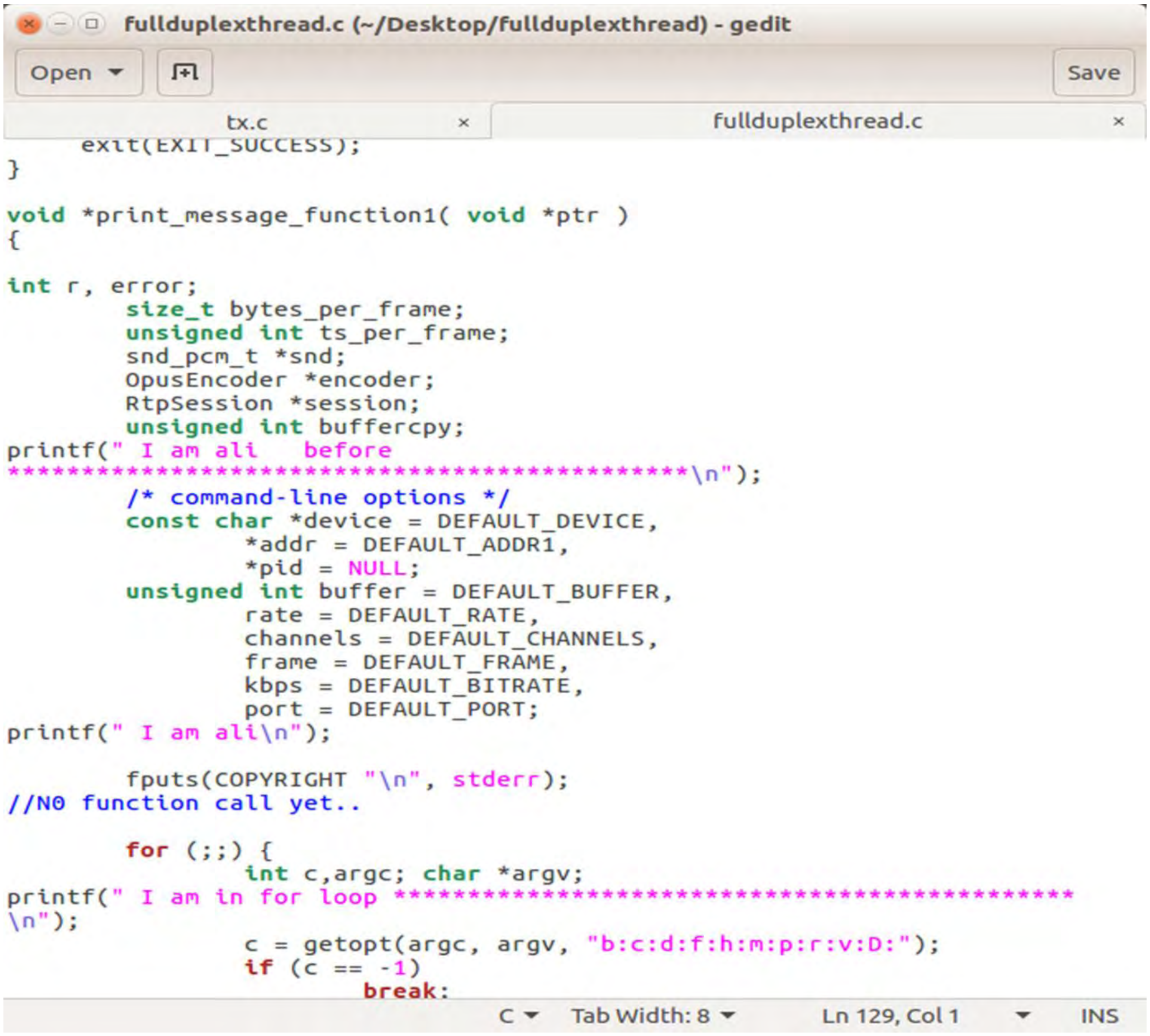
```
        return 0;
}

static int run_tx(snd_pcm_t *snd,
                const unsigned int channels,
                const snd_pcm_uframes_t frame,
                OpusEncoder *encoder,
                const size_t bytes_per_frame,
                const unsigned int ts_per_frame,
                RtpSession *session)
{
printf(" I am in run tx function with encoder
**********************************************\n");
        for (;;) {
                int r;

                r = send_one_frame(snd, channels, frame,
                                encoder, bytes_per_frame, ts_per_frame,
                                session);
                if (r == -1)
                        return -1;

                if (verbose > 1)
                        fputc('>', stderr);
        }
}

static void usage(FILE *fd)
{
        fprintf(fd, "Usage: tx [<parameters>]\n"
                "Real-time Full duplex audio over IP\n");

        fprintf(fd, "\nAudio device (ALSA) parameters:\n");
        fprintf(fd, "  -d <dev>    Device name (default '%s')\n",
                DEFAULT_DEVICE);
        fprintf(fd, "  -m <ms>     Buffer time (default %d milliseconds)\n"
```

Figure 3.9: Full duplex threading (g)



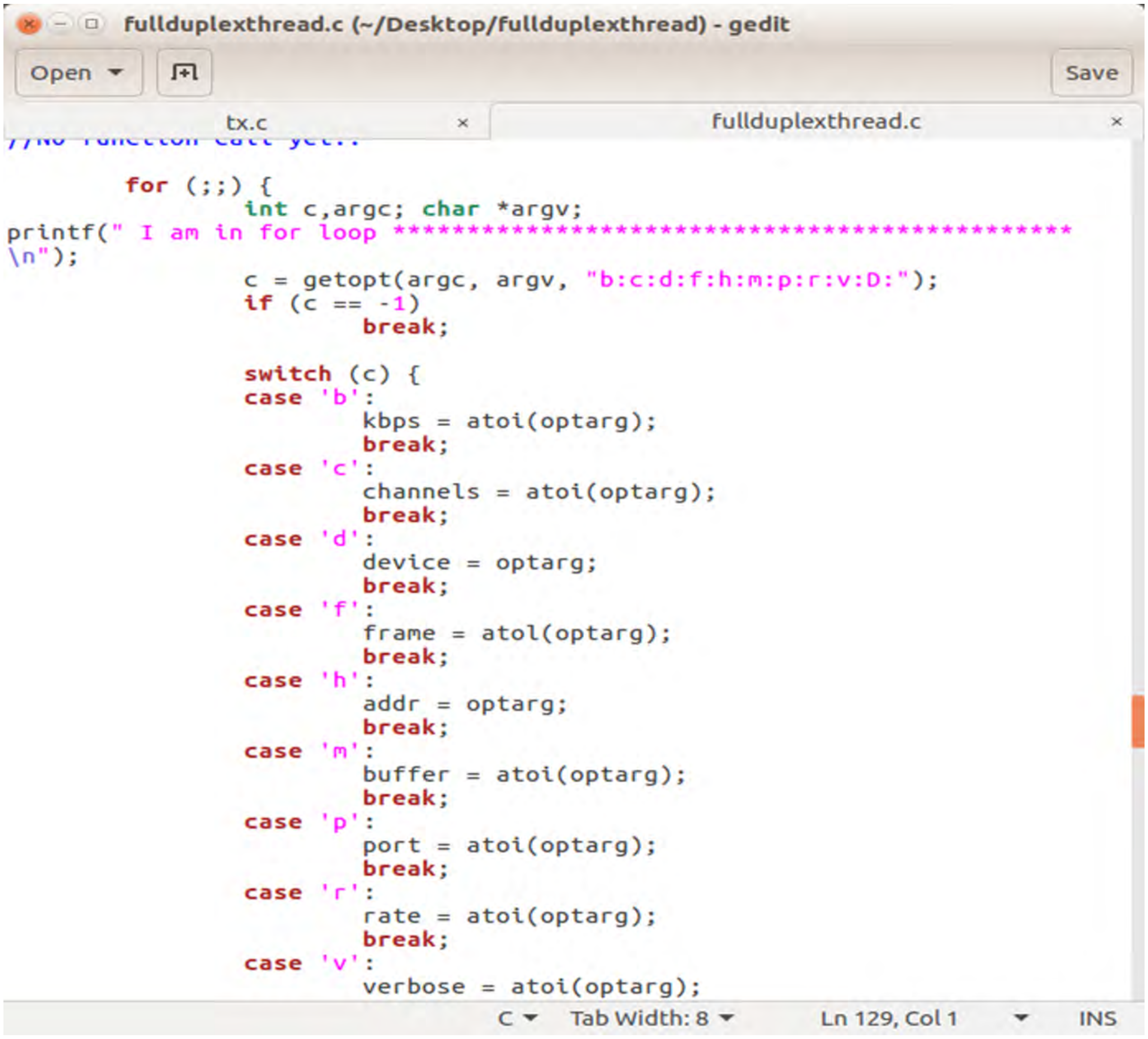
```
        fprintf(fd, "  -d <dev>     Device name (default '%s')\n",
                DEFAULT_DEVICE);
        fprintf(fd, "  -m <ms>      Buffer time (default %d milliseconds)\n",
                DEFAULT_BUFFER);

        fprintf(fd, "\nNetwork parameters:\n");
        fprintf(fd, "  -h <addr>    IP address to send to (default %s)\n",
                DEFAULT_ADDR1);
        fprintf(fd, "  -p <port>    UDP port number (default %d)\n",
                DEFAULT_PORT);
        printf("-j parameter is only for rx\n");
        fprintf(fd, "  -j <ms>      Jitter buffer (default %d milliseconds)
\n",
                DEFAULT_JITTER);

        fprintf(fd, "\nEncoding parameters:\n");
        fprintf(fd, "  -r <rate>    Sample rate (default %dHz)\n",
                DEFAULT_RATE);
        fprintf(fd, "  -c <n>       Number of channels (default %d)\n",
                DEFAULT_CHANNELS);
        fprintf(fd, "  -f <n>       Frame size (default %d samples, see
below)\n",
                DEFAULT_FRAME);
        fprintf(fd, "  -b <kbps>    Bitrate (approx., default %d)\n",
                DEFAULT_BITRATE);

        fprintf(fd, "\nProgram parameters:\n");
        fprintf(fd, "  -v <n>       Verbosity level (default %d)\n",
                DEFAULT_VERBOSE);
        fprintf(fd, "  -D <file>    Run as a daemon, writing process ID to
the given file\n");

        fprintf(fd, "\nAllowed frame sizes (-f) are defined by the Opus
codec. For example,\n"
                "at 48000Hz the permitted values are 120, 240, 480 or 960.
\n");
```

Figure 3.9: Full duplex threading (i)



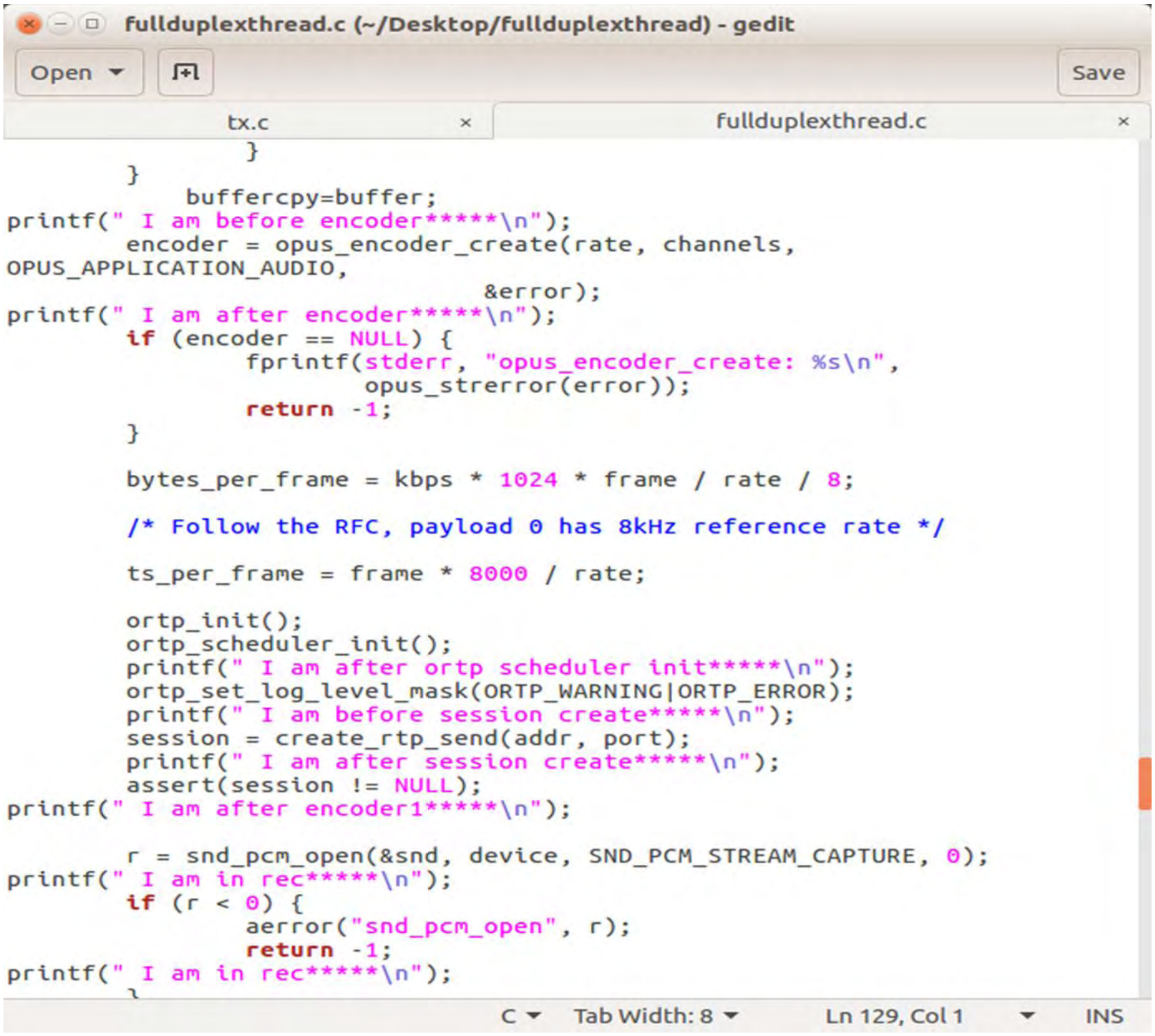
```
    exit(EXIT_SUCCESS);
}

void *print_message_function1( void *ptr )
{

int r, error;
        size_t bytes_per_frame;
        unsigned int ts_per_frame;
        snd_pcm_t *snd;
        OpusEncoder *encoder;
        RtpSession *session;
        unsigned int buffercpy;
printf(" I am ali   before
**********************************************\n");
        /* command-line options */
        const char *device = DEFAULT_DEVICE,
                *addr = DEFAULT_ADDR1,
                *pid = NULL;
        unsigned int buffer = DEFAULT_BUFFER,
                rate = DEFAULT_RATE,
                channels = DEFAULT_CHANNELS,
                frame = DEFAULT_FRAME,
                kbps = DEFAULT_BITRATE,
                port = DEFAULT_PORT;
printf(" I am ali\n");

        fputs(COPYRIGHT "\n", stderr);
//N0 function call yet..

        for (;;) {
                int c,argc; char *argv;
printf(" I am in for loop **********************************************
\n");
                c = getopt(argc, argv, "b:c:d:f:h:m:p:r:v:D:");
                if (c == -1)
                        break;
```

Figure 3.9: Full duplex threading (j)


fullduplexthread.c (~/Desktop/fullduplexthread) - gedit

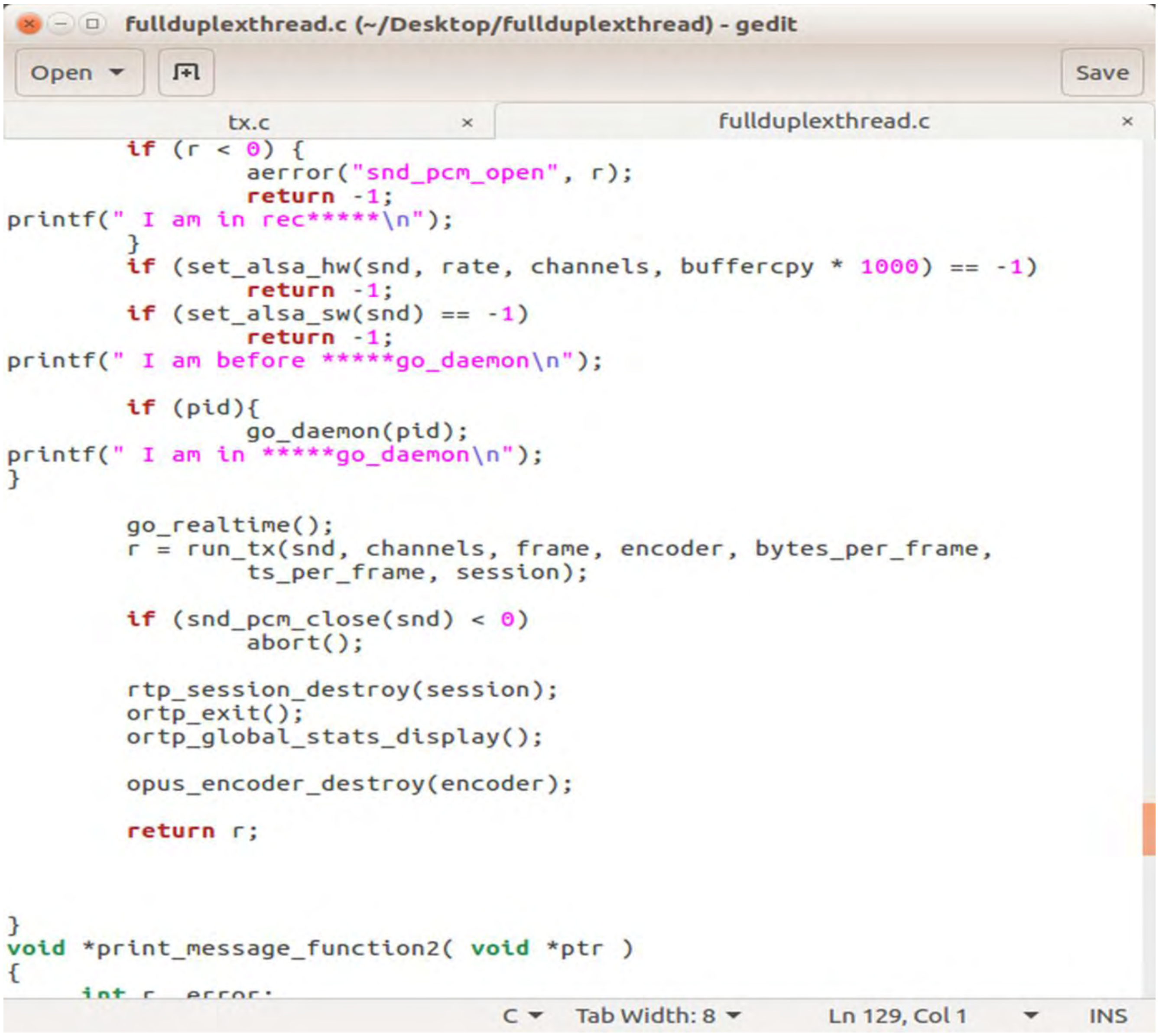
```
        for (;;) {
                int c,argc; char *argv;
printf(" I am in for loop ****************************************************
\n");
                c = getopt(argc, argv, "b:c:d:f:h:m:p:r:v:D:");
                if (c == -1)
                        break;

                switch (c) {
                case 'b':
                        kbps = atoi(optarg);
                        break;
                case 'c':
                        channels = atoi(optarg);
                        break;
                case 'd':
                        device = optarg;
                        break;
                case 'f':
                        frame = atol(optarg);
                        break;
                case 'h':
                        addr = optarg;
                        break;
                case 'm':
                        buffer = atoi(optarg);
                        break;
                case 'p':
                        port = atoi(optarg);
                        break;
                case 'r':
                        rate = atoi(optarg);
                        break;
                case 'v':
                        verbose = atoi(optarg);
```

Figure 3.9: Full duplex threading (l)


fullduplexthread.c (~/Desktop/fullduplexthread) - gedit

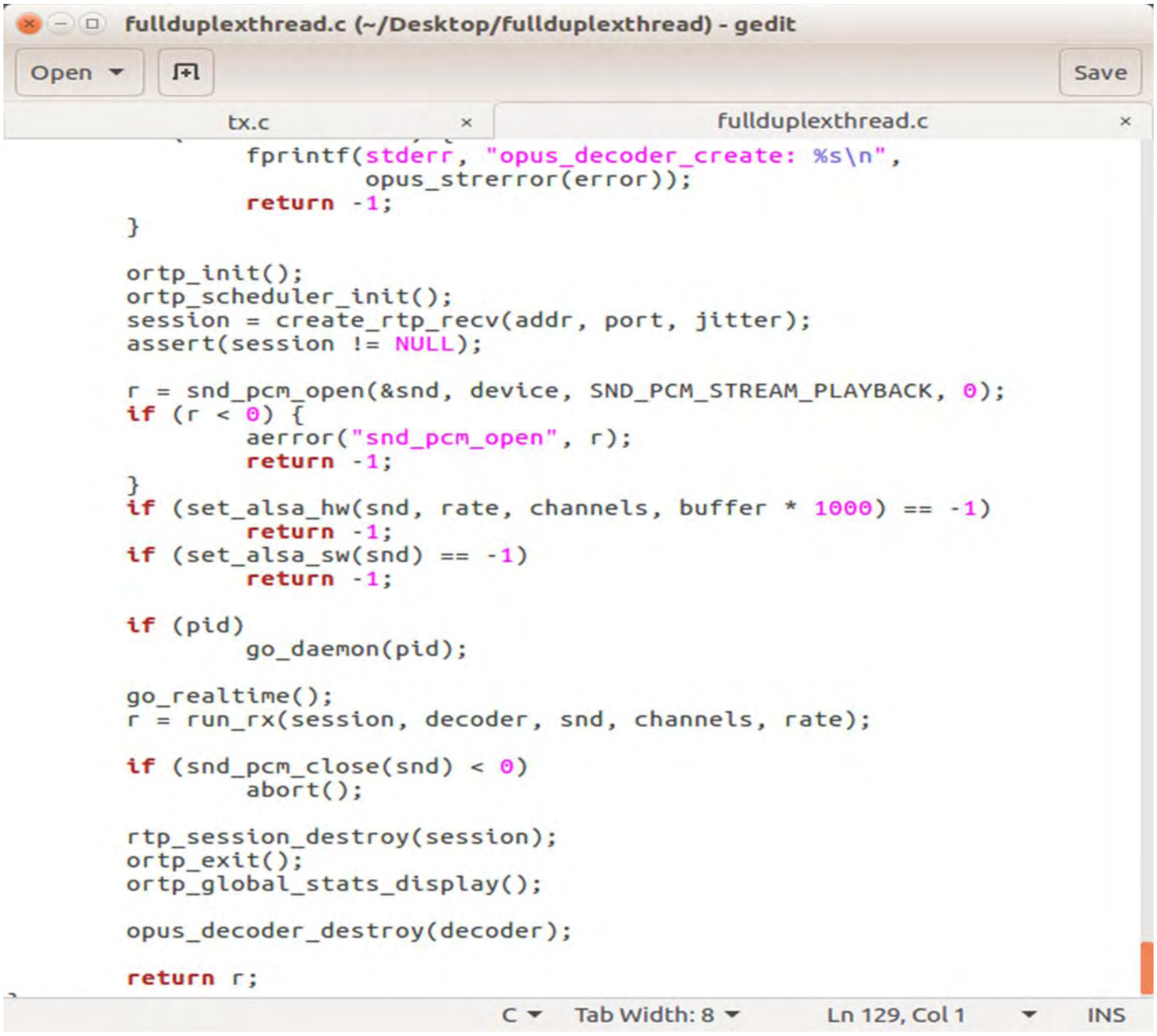
```
                }
        }
            buffercpy=buffer;
printf(" I am before encoder*****\n");
        encoder = opus_encoder_create(rate, channels,
OPUS_APPLICATION_AUDIO,
                                        &error);
printf(" I am after encoder*****\n");
        if (encoder == NULL) {
                fprintf(stderr, "opus_encoder_create: %s\n",
                        opus_strerror(error));
                return -1;
        }

        bytes_per_frame = kbps * 1024 * frame / rate / 8;

        /* Follow the RFC, payload 0 has 8kHz reference rate */

        ts_per_frame = frame * 8000 / rate;

        ortp_init();
        ortp_scheduler_init();
        printf(" I am after ortp scheduler init*****\n");
        ortp_set_log_level_mask(ORTP_WARNING|ORTP_ERROR);
        printf(" I am before session create*****\n");
        session = create_rtp_send(addr, port);
        printf(" I am after session create*****\n");
        assert(session != NULL);
printf(" I am after encoder1*****\n");

        r = snd_pcm_open(&snd, device, SND_PCM_STREAM_CAPTURE, 0);
printf(" I am in rec*****\n");
        if (r < 0) {
                aerror("snd_pcm_open", r);
                return -1;
printf(" I am in rec*****\n");
```

Figure 3.9: Full duplex threading (m)


fullduplexthread.c (~/Desktop/fullduplexthread) - gedit

```
        if (r < 0) {
                aerror("snd_pcm_open", r);
                return -1;
printf(" I am in rec*****\n");
        }
        if (set_alsa_hw(snd, rate, channels, buffercpy * 1000) == -1)
                return -1;
        if (set_alsa_sw(snd) == -1)
                return -1;
printf(" I am before *****go_daemon\n");

        if (pid){
                go_daemon(pid);
printf(" I am in *****go_daemon\n");
}

        go_realtime();
        r = run_tx(snd, channels, frame, encoder, bytes_per_frame,
                ts_per_frame, session);

        if (snd_pcm_close(snd) < 0)
                abort();

        rtp_session_destroy(session);
        ortp_exit();
        ortp_global_stats_display();

        opus_encoder_destroy(encoder);

        return r;



}
void *print_message_function2( void *ptr )
{
    int r, error;
```

Figure 3.9: Full duplex threading (n)



fullduplexthread.c (~/Desktop/fullduplexthread) - gedit

```
                fprintf(stderr, "opus_decoder_create: %s\n",
                        opus_strerror(error));
                return -1;
        }

        ortp_init();
        ortp_scheduler_init();
        session = create_rtp_recv(addr, port, jitter);
        assert(session != NULL);

        r = snd_pcm_open(&snd, device, SND_PCM_STREAM_PLAYBACK, 0);
        if (r < 0) {
                aerror("snd_pcm_open", r);
                return -1;
        }
        if (set_alsa_hw(snd, rate, channels, buffer * 1000) == -1)
                return -1;
        if (set_alsa_sw(snd) == -1)
                return -1;

        if (pid)
                go_daemon(pid);

        go_realtime();
        r = run_rx(session, decoder, snd, channels, rate);

        if (snd_pcm_close(snd) < 0)
                abort();

        rtp_session_destroy(session);
        ortp_exit();
        ortp_global_stats_display();

        opus_decoder_destroy(decoder);

        return r;
```

Figure 3.9: Full duplex threading (o)

3.4 ERRORS:

The errors which occur while running VoIP source code.

- Sched-setscheduler.
- Under-run Occurred.
- Over-run Occurred.

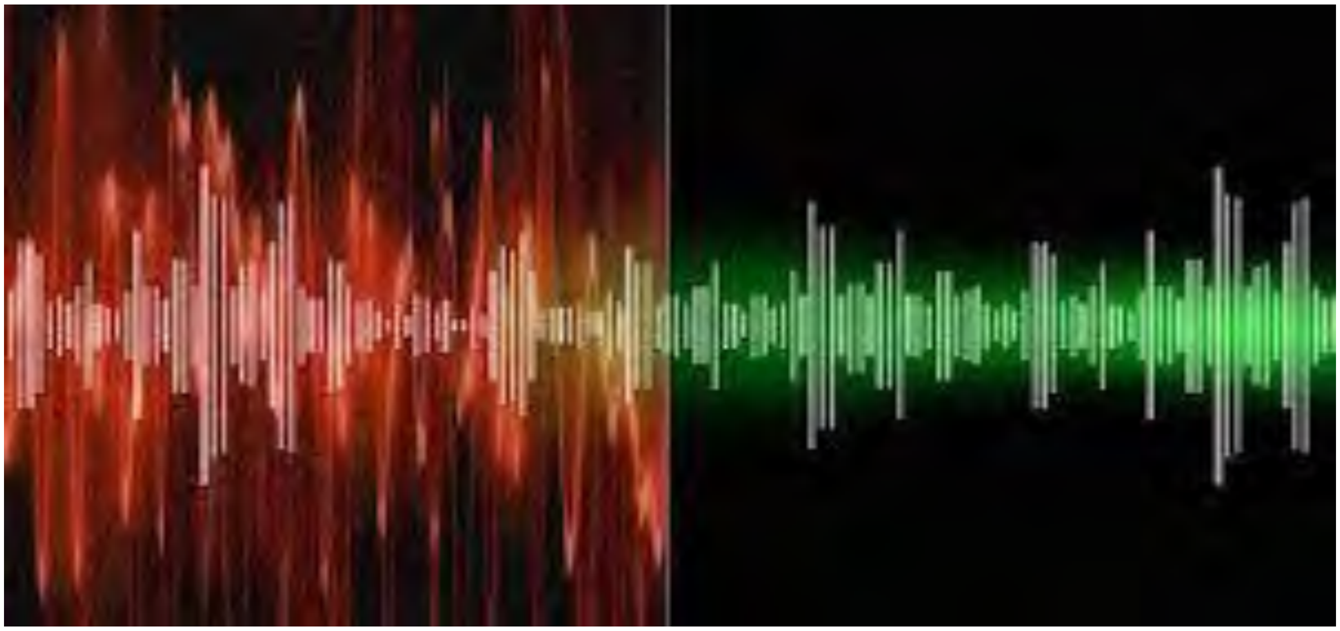
Sched-setscheduler function is used to set scheduling policy and its parameters for the process which is specified by Process ID (pid) and for the real time processing Scheduler allows the task to process and sets its priority at high, to set its priority at high user permission required, if permission not granted it gives the message as output permission not granted. To root the system it is necessary to type 'sudo' before run the program.

While Playback the audio if software application does not pass the data rapidly within required time into the buffer the resulting error called under-run and when device is ready for the data transmission continuously between hardware and software application buffers. While capturing the data if the application does not read the given data in the buffer quickly, the circular buffer will be overwrite with a totally new data. Because of lagging application the data will be loss, this result called over-run. If Under-run and Over-run message occurred it indicates the issues but by using recover function it recovers the lost data with a very low latency which is not noticeable.

CHAPTER 4

NOISE REDUCTION

Noise suppression is a method of eliminating unwanted distortion from a desired signal.



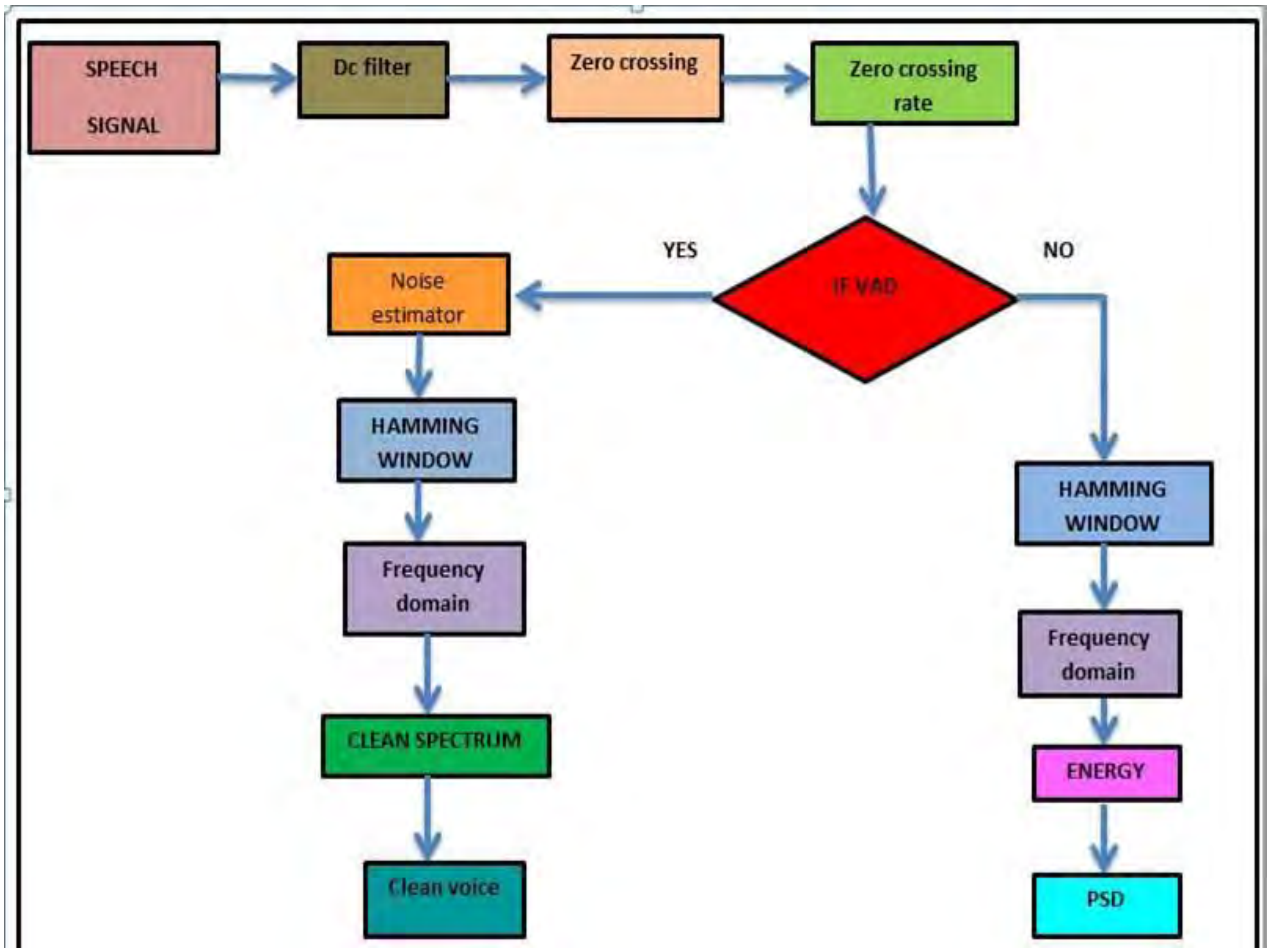
NOISE:

An unwanted distortion which create the annoying effect in signal.

4.1 SPEECH ENHANCEMENT OBJECTIVES

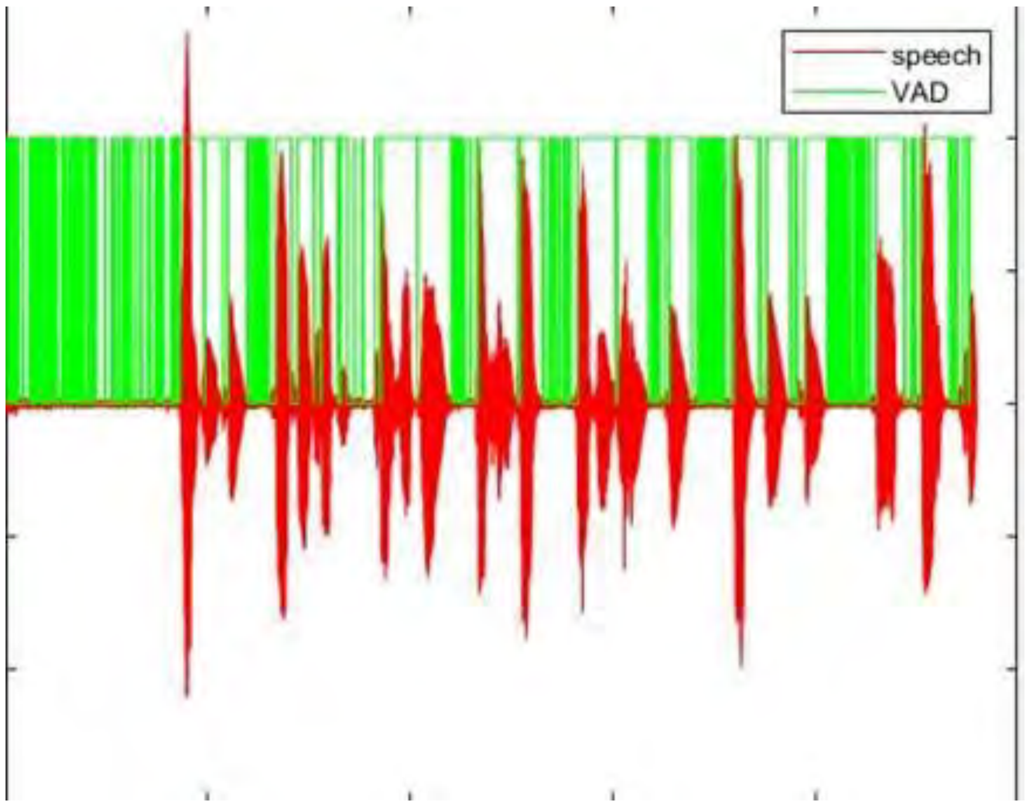
- Remove fair amount of Noise
- VAD
- Noise Estimation
- Spectral Subtraction
- Clean Speech
- Smoothing
- Amplification

4.2 BASIC FLOW CHART FOR NOISE REDUCTION ALGORITHM



4.3 TECHNIQUES

- **Zero-Crossing Detector:**
  Zero Crossing detectors is used to detect the signal wave transition from Positive to negative. Using the rate of zero crossing we can design VAD. If Zero Crossing rate is high means there is Noise. If Zero Crossing rate is low means there is Speech
- **Voice Active Detector:**
  The name implies itself that it is a Voice active detector. It is useful tool to design Noise Reduction application. It detects the Speech and Non-Speech activity in a Signal.



4.4 BENEFITS OF PROJECT

Noise suppression is the method of suppression the noise from desired signal in order to enhance the Speech quality in Audio Conferencing. The main objective is to suppress the Noise during Audio Conferencing. Noise Suppression technique effects the transmitted sound quality. For Example

The background noise which comes from both sides. Noise suppression filters it out for both sides. Many algorithms were designed to improve the sound quality.
Existing Noise Suppression solutions are not perfect to completely filter out the Speech form noisy environment. By the use of noise suppression algorithm it allows the receiver to hear the voice clearly. It benefits the user of Audio Conferencing and improves their ability to hear more clearly. Our aim is to provide high level of audio enhancement comfort during Audio Conferencing using Noise suppression technique.

4.5 FUTURE EVALUATION AND CONCLUSION

FUTURE EVALUATION

A server can be added to the application to register multiple clients, which can make the proposed system a simple VoIP application.

CONCLUSION

This thesis described a detailed discussion on Centralized Speech Enhancement for Audio Conferencing using VoIP services. We have focused on Speech Enhancement, in order to provide good Quality of services while using VoIP on Linux System. VoIP quality depends on many factors related to network conditions in order to provide Quality voice also controls the utilization of available resources. It is designed for real time Audio Communication over IP for both half duplex and full duplex communication. This thesis is written to provide the thorough knowledge about the Centralized Speech Enhancement for Audio Conferencing using VoIP and its Services.