電気電子情報実験・演習第一 I3 実験レポート

電子情報工学科
03240403 井上聡士

2024 年 6 月 28 日

1　目標

普通の通話アプリ (Zoom、Skype 等) には、以下のような機能が存在する

- 音声通話
- 多人数通話
- マイクのミュート
- カメラのオン・オフ
- 画面共有
- チャット

このうち、カメラ関係の機能は、WSL2 においてカメラのインターフェイスがどうなっているのかわからないことにより断念した。そのため、I3 課題では以下の機能を実装することを目標とした。

- 多人数通話
- ニックネームの設定
- マイクのミュート・アンミュート
- チャット

2　多人数通話

多人数通話をするにあたって、問題点が 4 点存在する。

- そもそもどのように通信を行うか
- 単位時間に送信するデータ量と受信するデータ量が大きく異なり、人数に依存する中どのようにデータの送受信を行うか
- 音声、チャット等様々なデータをどう区別するか
- 受信した音声をどのように再生すれば重ねることが出来るか

2.1 通信方式

そもそも基本的にサーバー・クライエントは 1 対 1 で通信を行うため、多人数通話を行うにはサーバー・クライエントの通信を複数行う必要がある。ただし、クライエント (=アプリのユーザー) が複数の他のクライエントに接続するのは、非常に難しく、そもそも通信に負荷がかかるうえ、さらにクライエント同士で接続する際にどちらがサーバーとなるかわからないという問題も存在する。そのため、サーバー側で複数のクライエントを処理し、それぞれのクライエントに複数のクライエント分のデータを送ることで、多人数通話を実現することにした (図 1)。サーバーは基本的にデータの仲介のみを行い、それ以外の機能を担わないが、例外として新しいクライエントが接続してきたときに他のクライエントの現在の状況 (ニックネーム、ミュート状態) を送る。クライエントで多くの処理を行い、音声、ミュート・アンミュート、チャットの情報を受け取った後にそれを処理する。

さて、既存のクライエントや新しい接続を試みるクライエントが存在するとき、どのように処理するべきか。通常の recv 関数は、基本的に相手から信号を受信するまで待機し続け、その間に他の処理を実行することが出来ない。そこで、select という関数を使用した。select 関数は、複数の file descriptor に対して、読み込み、書き込み、例外のいずれかが発生するまで待機する関数であり、逆にいえばどれか一つが変化すれば処理を行うことが出来る。基本的な使用方法は、以下の通りである。

```
fd_set readfds;
int maxfd = 0;
while (1) {
    FD_ZERO(&readfds); // readfdsを初期化
    FD_SET(fd0, &readfds); // 全てのfile descriptorをreadfdsに追加
    ...
    FD_SET(fdN, &readfds);
    maxfd = max(maxfd, fd0, fd1, ..., fdN); // 最大のfile descriptorを取得
    select(maxfd + 1, &readfds, NULL, NULL, NULL); // いずれかのfile descriptorが変化するまで
    待機
    if (FD_ISSET(fd0, &readfds)) {
        // fd0に変化があった場合の処理
    }
    ...
    if (FD_ISSET(fdN, &readfds)) {
        // fdNに変化があった場合の処理
    }
}
```

実際のサーバーのスクリプトでは、新規クライエントからの接続はサーバーの socket の変化を、既存のクライエントからのデータは各クライエントの socket の変化を検知することで処理した。

2.2 クライエント側のデータ処理

クライエント側は、通話に参加しているユーザーの数により受信するデータ量が変化するため、I2 で行ったようにデータを受信・送信交互に行っても遅延が生じてしまう。そのため、ノンブロッキングモードで入力した音声データの送信と受信したサーバーデータの処理を行う必要がある。

この処理も、select 関数を使用して行った。connect したサーバーの socket と、起動した sox の

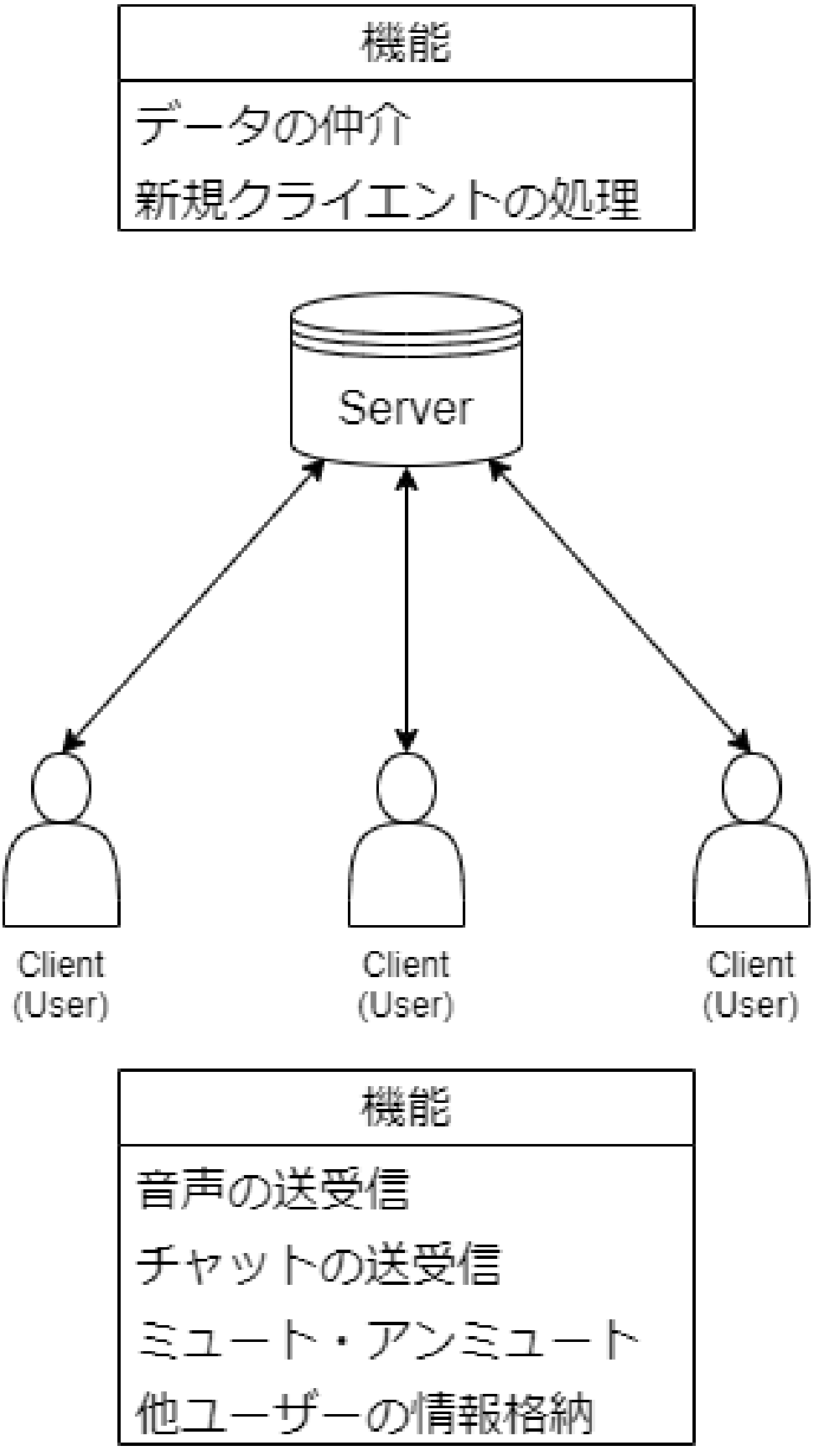

図 1: 通信方式の設計と機能

socket(および後述するが標準入力 stdin の socket である STDIN_FILENO) を select 関数に登録し、どちらかに変化があった場合に処理するようにした。

2.3 データの区別

サーバーからは、自分以外のクライエント分の音声データ、チャットデータ、及びクライエントの状態 (ミュート・アンミュート、ニックネーム) 変化のデータが送られてくるが、これを区別する必要がある。そこで、サーバーへの送信データおよびサーバーからの受信データの先頭に、クライエント番号とデータの種類を示す識別番号を付与することで、クライエント側でデータの種類を判別することにした。サーバーへ送信するときは、識別番号をそれぞれ以下の通りに設定した

- 1: マイクからの音声データ
- 2: チャットのデータ
- 3: ミュート・アンミュートのデータ
- 4: ニックネーム・表示名のデータ

サーバーから受信するときは、識別番号をそれぞれ以下の通りに設定した。

- 0: サーバーからのメッセージ (主にサーバーへの接続で帰ってくるメッセージ)
- 1: 新規クライエントの接続・通話への参加
- 2: 他のクライエントの切断・通話からの離脱
- 3: 音声データ
- 4: チャットのデータ
- 5: ミュート・アンミュートの変更
- 6: ニックネーム・表示名の変更

そして、各 TCP パケットの先頭に、サーバーへの送信データは **(識別番号)**>>を、サーバーからの受信データは **(クライエント番号).(識別番号)**>>をつけて送信した。

しかし、ここで 3 つほど問題点が浮上した。

- 付けた修飾子...>>の文字数が不定で、どこまで読み取ればいいのか不明である
- 修飾子...>>が音声データやチャットデータにも含まれてしまうと、音声データが途中で切れたりしてしまい、挙動が不安定になる
- recv で特定の文字数を取得しようとしても、それより短い文字数しか取得できない場合がある

上の二つは、どちらも「recv する文字数が不明である」という共通の問題点に起因している。これを解決するために、まず、修飾子...>>の文字数を統一した。接続するクライエントは最大 30 に設定してあったため、クライエント番号を 2 桁表記 %02d にすることで解決した。次に、2 番目の問題だが、これは事前に読み取るデータの長さを指定することで回避できる。音声データは 1024 バイトずつ送信することに決めていたので、1 回のパケットで送る最大データ量を 1024 バイト + 修飾子として、その送信するデータ量を修飾子の中に記述した。例えば、140 バイトの文字列をサーバーへ送信する場合、2.0140>>と記述した後に 140 字の文字列を送信することで、サーバー側で 140 バイトの文字列として処理でき、仮に 2.0300>>などの文字列がデータ中に現れても問題なく文字として処理できる。

最後に、3 番目の問題は、検証した結果、TCP パケットをすべて受信できる前に recv を呼び出したことによるものだと判明した。通常 TCP パケットはデータ量が大きいと何回かに分割して送るが、そのうち一つしか受信できてないうちに recv を呼ぶと意図していたよりも短いデータしか読み取れない。これを修正するために、パケットのすべてのデータを受け取るまで待つ recv_all という関数を定義した。

2.4 音声の入力・出力

音声の入力・出力には、sox を使用した。ただし、チャット機能を実装するが故、標準入力は必要であったので、popen を使用して独自の file descriptor から入力を取ることで、sox と標準入力を同時に行うことが出来るようにした。

出力は、これより難易度が高かった。sox のマニュアルを見ると、複数の入力を同時に再生することが可能で、かつ-v をつけることでそれぞれの入力の音量を独自に調整できるようなので、これを試みた [1]。そこで、クライエント側で、他の各クライエントの音声入力を収納する一時ファイル tmp01.tmp、…tmpN.tmp を作成し、それを

```
$ sox -m -q -t s16 -r 44100 -c 1 tmp01.tmp -t s16 -r 33100 -c 1 tmp02.tmp ... -d
```

として、popen で実行するようにした。また、新しいクライエントが入ったときは元の popen を pclose で閉じて、新しいものを開くようにした。しかし、これはうまく動作しなかった。音声は重なったが、一定の時間再生された後再生されなくなった。これは、sox が、起動された時までの.tmp ファイルの分の時間しか再生しなかったためであると考えられる。

そこで、この解決策、すなわち popen を起動した後のファイルの更新を読み取る仕組みを考えた。このとき、2 通りの似たような方法にたどり着いた。

一つ目は、tail -f コマンドによる追跡である [2]。tail -f コマンドは、あるファイルを追跡し、随時更新されるたびに新たな出力を生み出し、かつキャンセルされるまで終了しない関数である。sox コマンドは、パイプのような形で複数のコマンド出力を取ることも可能であり、以下のように記述される。

```
$ sox -m -q -t s16 -r 44100 -c 1 "|tail -f tmp01.tmp" -t s16 ...
```

これを popen で実行することで、sox が tmp01.tmp、tmp02.tmp、... の更新を追跡し、新たな音声データが追加されるたびに再生することが出来る。しかし、新たなクライエントが登場し、pclose でこのコマンドを閉じようとしても閉じず、そこでフリーズするという問題点が発生した。これは、pclose の特性上、実行しているプログラムを kill のように強制終了するのではなく、終了するまで待つためである。しかし、sox によりパイプされた tail -f は、おそらくサブプロセスとして実行され、sox が終了しても tail -f は終了しないため、pclose が終了しないと考えられる。

二つ目は、FIFO(First in, first out) という種類のファイルを使用することである。FIFO は、ファイルシステム上に存在するファイルであり、書き込まれたデータは読み取られるまで保持される [4]。外見上は他の普通のファイルと変わりはないが、ls -l コマンドで p という文字が先頭についていることで FIFO であることがわかる。FIFO は、mkfifo コマンドで作成することが出来る。また、この FIFO は、ファイルの書き込みと読み込みを同時に開く必要があり、同時開かれるまで open の行で止まったままになってしまう。そのため、open を別スレッドで行うことで、FIFO を開こうとした。fork 関数を使用して、まず親プロセスで sox を起

動し、子プロセスで FIFO(読み取り) を開き、その後親プロセスで open(書き込み) をすることで、FIFO を同時に開くことが出来た。しかし、これもうまく動作しなかった。検証を重ねていくうちに、おそらく sox コマンドが複数の FIFO の入力に対応していないことが判明した。

そのため、最終的には、一つの sox コマンドですべてのクライエントの音声を再生することは不可能であると判断した。そこで、複数の sox コマンドを起動し、それぞれのクライエントの音声を再生することにした。

この方針は簡単で、popen により複数のコマンドを同時に起動し、それぞれの入力 file descriptor に書き込むことで実現できた。

3 チャット

チャットはいたって簡単である。クライエント側の select で、標準入力も監視し、変更があった際に識別子を付けてサーバーに送信する。受信する際は、読み取った識別子からチャットであることを判断し、それを標準出力に出力する。

4 コマンド

ミュート・アンミュート、および名前の変更は、標準入力で/mute、/unmute、/name **<名前>**と入力することで行うことが出来る。標準入力のデータを、strncmp で比較し、最初の文字が/であればコマンドと認識し、それ以降の文字列を処理する。

ミュートをしたときには、音声データを単に送信しないようにした。

5 コード

5.1 実行

コンパイルしたサーバー側のファイルを以下のように実行する。

```
$ ./multi_server <ポート番号>
```

別のターミナルで、コンパイルしたクライエント側のファイルを以下のように実行する。

```
    $ ./multi_client <サーバーのIPアドレス> <ポート番号>
```

5.2 実演

レポートを書く際は 4 つのターミナルを用意し、サーバーを起動した後 3 つのクライエントを起動した。尚、別の PC で実行しても動作することは確認済みである。図 2 は、この時の様子である。サーバーを起動した後、左側のクライエントから順に起動した。新しいクライエント (左から 2 つめの Client 5) が参加すると、既存の Client 4 には Client 5>> Connected と表示され、接続されたことが表示される。また、参加した Client 5 も、すでにいる Client 4 の情報を受け取っている。

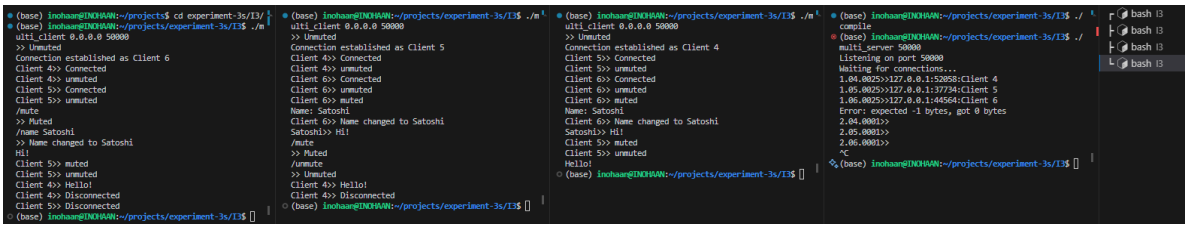

図 2: 実演した際の様子

最後に参加した Client 6 が/mute コマンドを使用すると、Client 6>> muted と表示され、Client 6 の音声が送信されなくなる。また、name Satoshi とした後は、Client 6>>の代わりに Satoshi>>と表示されている。Hi!と入力すると他のクライエントにも Satoshi>> Hi!と表示されているように、チャットも機能している。

また、クライエントが切断すると、Client 4>> Disconnected のように、切断も表示される。

レポートで紹介した機能は正常に動作しているだろう。

5.3 サーバー側のコード

```c
#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/select.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define BUF_READ 1024
#define BUF_SEND 1060
#define MAX_CLIENTS 30

// Global
int client_sockets[MAX_CLIENTS];
int muted[MAX_CLIENTS];
char usernames[MAX_CLIENTS][21];
int ss;

int send_all(int *client_sockets, int client_exclude, char *msg, int len);
int recv_all(int socket, char *buffer, int length);
int buf_w_bytes(char *sendbuf, int *len, int identifier, int sd, char *buf);
int buf_w_str(char *sendbuf, int *len, int identifier, int sd, char *str);
int handle_sigint(int sig);

int main(int argc, char **argv) {
    int port;
    int addrlen, max_sd, sd, len_buf, recv_identifier;
    struct sockaddr_in addr;
    fd_set readfds;
    char readbuf[BUF_READ];
    char sendbuf[BUF_SEND];
    char ipbuffer[30];
    for (int i = 0; i < MAX_CLIENTS; i++) {
        client_sockets[i] = 0;
    }

    if (argc < 2) {
        perror("Usage: ./multi_server <port>");
        exit(EXIT_FAILURE);
    }
    // Server
    if ((port = atoi(argv[1])) == 0) {
        perror("port");
        exit(EXIT_FAILURE);
    }
    if ((ss = socket(PF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port = htons(port);
```

```
    if (bind(ss, (struct sockaddr *)&addr, sizeof(addr)) == -1) {
        perror("bind");
        exit(EXIT_FAILURE);
    }
    printf("Listening on port %d\n", port);
    if (listen(ss, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    addrlen = sizeof(addr);
    printf("Waiting for connections...\n");

    while (1) {
        FD_ZERO(&readfds);
        FD_SET(ss, &readfds);
        max_sd = ss;
        for (int i = 0; i < MAX_CLIENTS; i++) {
            sd = client_sockets[i];
            if (sd > 0) {
                FD_SET(sd, &readfds);
            }
            if (sd > max_sd) {
                max_sd = sd;
            }
        }
        int activity = select(max_sd + 1, &readfds, NULL, NULL, NULL);
        if ((activity < 0) && (errno != EINTR)) {
            perror("select");
        }

        // Incoming connection
        if (FD_ISSET(ss, &readfds)) {
            if ((sd = accept(ss, (struct sockaddr *)&addr, (socklen_t *)&addrlen)) < 0) {
                perror("accept");
                exit(EXIT_FAILURE);
            }
            // Send to preexisting clients the information of newly connected client
            snprintf(readbuf, BUF_READ, "%s:%d:Client %d", inet_ntoa(addr.sin_addr), ntohs(addr.sin_port), sd);
            buf_w_str(sendbuf, &len_buf, 1, sd, readbuf);
            puts(sendbuf);
            if (send_all(client_sockets, sd, sendbuf, len_buf) < 0) {
                perror("send_all");
            }
            snprintf(readbuf, BUF_READ, "%d", sd);
            buf_w_str(sendbuf, &len_buf, 0, ss, readbuf);
            if (send(sd, sendbuf, len_buf, 0) != len_buf) {
                perror("send");
            }
            int is_registered = 0;
            // Add new socket to array of sockets
            for (int i = 0; i < MAX_CLIENTS; i++) {
                if (client_sockets[i] == 0 && !is_registered) {
                    client_sockets[i] = sd;
                    muted[i] = 0;
                    usernames[i][0] = '\0';
                    is_registered = 1;
                } else if (client_sockets[i] != 0) {  // Send old clients to new client
                    getpeername(client_sockets[i], (struct sockaddr *)&addr, (socklen_t *)&addrlen);
                    snprintf(readbuf, BUF_READ, "%s:%d", inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));
                    buf_w_str(sendbuf, &len_buf, 1, client_sockets[i], readbuf);
                    if (send(sd, sendbuf, len_buf, 0) != len_buf) {
                        perror("send_all");
                    }
                    // Mute status
                    len_buf = sizeof(char);
                    buf_w_bytes(sendbuf, &len_buf, 5, client_sockets[i], (char *)&muted[i]);
                    if (send(sd, sendbuf, len_buf, 0) != len_buf) {
                        perror("send_all");
                    }
                    // Username
                    if (strcmp(usernames[i], "") != 0) {  // If not blank
                        buf_w_str(sendbuf, &len_buf, 6, client_sockets[i], usernames[i]);
                        if (send(sd, sendbuf, len_buf, 0) != len_buf) {
                            perror("send_all");
                        }
                    }
                }
            }
        }
    }
```

```
        // Other sockets (client)
        for (int i = 0; i < MAX_CLIENTS; i++) {
            sd = client_sockets[i];
            if (FD_ISSET(sd, &readfds)) {
                readbuf[8] = '\0';
                if ((len_buf = recv_all(sd, readbuf, 8)) == 0) {
                    // Disconnected
                    getpeername(sd, (struct sockaddr *)&addr, (socklen_t *)&addrlen);
                    buf_w_str(sendbuf, &len_buf, 2, sd, "");
                    puts(sendbuf);
                    if (send_all(client_sockets, sd, sendbuf, len_buf) < 0) {
                        perror("send_all");
                    }
                    close(sd);
                    client_sockets[i] = 0;
                    muted[i] = 0;
                    usernames[i][0] = '\0';
                } else {
                    int actual_scanned;
                    // Get identifier
                    sscanf(readbuf, "%d.%d>>", &recv_identifier, &len_buf);
                    if ((actual_scanned = recv_all(sd, readbuf, len_buf)) != len_buf) {
                        fprintf(stderr, "Error: expected %d bytes, got %d bytes\n", len_buf, actual_scanned);
                        continue;
                    }
                    switch (recv_identifier) {
                        case 1:  // Sound data
                            buf_w_bytes(sendbuf, &len_buf, 3, sd, readbuf);
                            if (send_all(client_sockets, sd, sendbuf, len_buf) < 0) {
                                perror("send_all");
                            }
                            break;
                        case 2:  // Message
                            buf_w_str(sendbuf, &len_buf, 4, sd, readbuf);
                            if (send_all(client_sockets, sd, sendbuf, len_buf) < 0) {
                                perror("send_all");
                            }
                            break;
                        case 3:  // Mute & unmute
                            muted[i] = (int)readbuf[0];
                            len_buf = sizeof(char);
                            buf_w_bytes(sendbuf, &len_buf, 5, sd, (char *)&muted[i]);
                            if (send_all(client_sockets, sd, sendbuf, len_buf) < 0) {
                                perror("send_all");
                            }
                            break;
                        case 4:  // Username change
                            strcpy(usernames[i], readbuf);
                            buf_w_str(sendbuf, &len_buf, 6, sd, usernames[i]);
                            if (send_all(client_sockets, sd, sendbuf, len_buf) < 0) {
                                perror("send_all");
                            }
                            break;
                        default:
                            break;
                    }
                }
            }
        }
    }
    close(ss);
    return 0;
}

int send_all(int *client_sockets, int client_exclude, char *buf, int len) {
    int flag = 0;
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (client_sockets[i] != 0 && client_sockets[i] != client_exclude) {
            if (send(client_sockets[i], buf, len, 0) != len) {
                fprintf(stderr, "send_all: %d\n", client_sockets[i]);
                flag = 1;
            }
        }
    }
    return flag;
```

```c
}

int recv_all(int socket, char *buffer, int length) {
    int total_received = 0;
    int bytes_left = length;
    int n;

    while (total_received < length) {
        n = recv(socket, buffer + total_received, bytes_left, 0);
        if (n <= 0) {
            // Handle error or disconnection
            return n;
        }
        total_received += n;
        bytes_left -= n;
    }

    return total_received;
}

int buf_w_bytes(char *sendbuf, int *len, int identifier, int sd, char *buf) {
    if (*len > BUF_READ) {
        fprintf(stderr, "%d bytes exceeds buffer size\n", *len);
        return -1;
    }
    snprintf(sendbuf, BUF_SEND, "%d.%02d.%04d>>", identifier, sd, *len);
    bcopy(buf, sendbuf + 11, *len);
    *len += 11;
    return 0;
}

int buf_w_str(char *sendbuf, int *len, int identifier, int sd, char *str) {
    if (strlen(str) + 1 > BUF_READ) {
        fprintf(stderr, "String %s exceeds buffer size\n", str);
        return -1;
    }
    snprintf(sendbuf, BUF_SEND, "%d.%02d.%04ld>>%s", identifier, sd, strlen(str) + 1, str);
    *len = 11 + strlen(str) + 1;
    return 0;
}

int handle_sigint(int sig) {
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (client_sockets[i] != 0) {
            close(client_sockets[i]);
        }
    }
    close(ss);
    exit(0);
}
```

multi_server.c

5.4 クライエント側のコード

```c
#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/select.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#define BUF_READ 1024
#define BUF_SEND 1060
#define MAX_CLIENTS 30
#define COMMAND_LENGTH 500

typedef struct clientsettings {
    int id;
    char ip[17];
    unsigned int port;
    char usrname[21];
    FILE *fp;
    unsigned int volume;
    int is_muted;
} ClientSettings;

// Global variables
ClientSettings *clients[MAX_CLIENTS];
int s;

// Function prototypes
ClientSettings *client_setup(int sd, char *str);
int client_free(int sd);
ClientSettings *client_get(int sd);
int recv_all(int socket, char *buffer, int
```

```
    length);
int buf_w_bytes(char *sendbuf, int *len, int identifier, char *buf);
int buf_w_str(char *sendbuf, int *len, int identifier, char *str);
void handle_sigint(int sig);

int main(int argc, char **argv) {
    signal(SIGINT, handle_sigint);
    if (argc < 3) {
        perror("Usage: ./multi_client <ip> <port>");
    }
    fd_set readfds;
    int port = atoi(argv[2]);
    char *ip = argv[1];
    FILE *fp_in;
    int max_sd, fd_in, len_buf;
    int scan_type, scan_client, scan_len;
    int is_muted = 0;
    char buf[BUF_READ];
    char sendbuf[BUF_SEND];
    char buf_identifier[12];
    buf_identifier[11] = '\0';

    if ((s = socket(PF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    if (inet_aton(ip, &addr.sin_addr) == 0) {
        perror("inet_aton");
        return 1;
    }
    addr.sin_port = htons(port);
    int ret = connect(s, (struct sockaddr *)&addr, sizeof(addr));
    if (ret == -1) {
        perror("connect");
        return 1;
    }

    if ((fp_in = popen("rec -t s16 -c 1 -r 44100 -q --buffer 1024 -", "r")) == NULL) {
        perror("popen");
        return 1;
    }
    fd_in = fileno(fp_in);

    // Send initial information (mute, user name) first
    len_buf = sizeof(char);
    buf_w_bytes(sendbuf, &len_buf, 3, (char *)&is_muted);
    send(s, sendbuf, len_buf, 0);
    fprintf(stderr, is_muted ? ">> Muted\n" : ">> Unmuted\n");
    for (int i = 0; i < argc; i++) {
        if (strcmp(argv[i], "-u") == 0) {
            if (strlen(argv[i + 1]) > 20) {
                fprintf(stderr, "Username too long\n");
            } else {
                buf_w_str(sendbuf, &len_buf, 4, argv[i + 1]);
                send(s, sendbuf, len_buf, 0);
                fprintf(stderr, ">> Name changed to %s\n", argv[i + 1]);
            }
            break;
        }
    }

    while (1) {
        FD_ZERO(&readfds);
        // Incoming data from server
        FD_SET(s, &readfds);
        // Incoming data from stdin
        FD_SET(STDIN_FILENO, &readfds);
        // Incoming data from sox (mic)
        FD_SET(fd_in, &readfds);
        if (s > STDIN_FILENO && s > fd_in) {
            max_sd = s;
        } else if (fd_in > STDIN_FILENO && fd_in > s) {
            max_sd = fd_in;
        } else {
            max_sd = STDIN_FILENO;
        }
        int activity = select(max_sd + 1, &readfds, NULL, NULL, NULL);
        if ((activity < 0) && (errno != EINTR)) {
            perror("select");
        }
        // Incoming data from server
        if (FD_ISSET(s, &readfds)) {
            int actual_scanned;
            if (recv_all(s, buf_identifier, 11) == 0) {
                fprintf(stderr, "Server disconnected\n");
                break;
            }
            sscanf(buf_identifier, "%d.%d.%d>>", &scan_type, &scan_client, &scan_len);
            if ((actual_scanned = recv_all(s, buf, scan_len)) != scan_len) {
                fprintf(stderr, "[%s]", buf_identifier);
```

```
                fprintf(stderr, "Error: expected %d bytes, got %d bytes\n", scan_len, actual_scanned);
            }
            switch (scan_type) {
                case 0:  // message from server
                    fprintf(stderr, "Connection established as Client %s\n", buf);
                    break;
                case 1:  // new client
                    client_setup(scan_client, buf);
                    fprintf(stderr, "%s>> Connected\n", client_get(scan_client)->usrname);
                    break;
                case 2:  // client disconnected
                    fprintf(stderr, "%s>> Disconnected\n", client_get(scan_client)->usrname);
                    client_free(scan_client);
                    break;
                case 3:  // client data
                    if (client_get(scan_client)->fp < 0) {
                        fprintf(stderr, "%s not ready\n", client_get(scan_client)->usrname);
                        break;
                    }
                    fwrite(buf, sizeof(char), scan_len, client_get(scan_client)->fp);
                    break;
                case 4:  // Client chat
                    fprintf(stderr, "%s>> %s", client_get(scan_client)->usrname, buf);
                    break;
                case 5:  // Mute change
                    client_get(scan_client)->is_muted = buf[0];
                    fprintf(stderr, "%s>> %s\n", client_get(scan_client)->usrname, client_get(scan_client)->is_muted ? "muted" : "unmuted");
                    break;
                case 6:  // Name change
                    strcpy(client_get(scan_client)->usrname, buf);
                    fprintf(stderr, "Name: %s\n", client_get(scan_client)->usrname);
                    if (strcmp(client_get(scan_client)->usrname, "") == 0) {
                        snprintf(client_get(scan_client)->usrname, 21, "Client %d", scan_client);
                    }
                    fprintf(stderr, "Client %d>> Name changed to %s\n", client_get(scan_client)->id, buf);
                    break;
                default:
                    break;
            }
        }
        // Input from STDIN
        if (FD_ISSET(STDIN_FILENO, &readfds)) {
            fgets(buf, BUF_READ, stdin);
            if (strncmp(buf, "/", 1) == 0) {  // Commands
                if (strncmp(buf, "/mute", 5) == 0) {
                    is_muted = 1;
                    len_buf = sizeof(char);
                    buf_w_bytes(sendbuf, &len_buf, 3, (char *)&is_muted);
                    send(s, sendbuf, len_buf, 0);
                    fprintf(stderr, ">> Muted\n");
                } else if (strncmp(buf, "/unmute", 7) == 0) {
                    is_muted = 0;
                    len_buf = sizeof(char);
                    buf_w_bytes(sendbuf, &len_buf, 3, (char *)&is_muted);
                    send(s, sendbuf, len_buf, 0);
                    fprintf(stderr, ">> Unmuted\n");
                } else if (strncmp(buf, "/name", 5) == 0) {
                    buf[strlen(buf) - 1] = '\0';
                    if (strlen(buf + 6) > 20) {
                        fprintf(stderr, "Name too long\n");
                    } else {
                        buf_w_str(sendbuf, &len_buf, 4, buf + 6);
                        send(s, sendbuf, len_buf, 0);
                        fprintf(stderr, ">> Name changed to %s\n", buf + 6);
                    }
                } else {
                    fprintf(stderr, "Unknown command\n");
                }
            } else {
```

```
                buf_w_str(sendbuf, &len_buf, 2, buf);
                send(s, sendbuf, len_buf, 0);
            }
        }

        // Input from sox (mic)
        if (FD_ISSET(fd_in, &readfds)) {
            if ((len_buf = read(fd_in, buf, BUF_READ)) > 0) {
                if (!is_muted) {
                    buf_w_bytes(sendbuf, &len_buf, 1, buf);
                    send(s, sendbuf, len_buf, 0);
                }
            }
        }
    }
    handle_sigint(SIGINT);
}

ClientSettings *client_setup(int sd, char *str) {
    ClientSettings *client = malloc(sizeof(ClientSettings));
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i] == NULL) {
            clients[i] = client;
            break;
        }
    }
    client->id = sd;
    char *token = strtok(str, ":");
    strcpy(client->ip, token);
    token = strtok(NULL, ":");
    client->port = atoi(token);
    snprintf(client->usrname, 21, "Client %d", client->id);
    client->volume = 100;
    client->is_muted = 0;
    if (client->fp != NULL) {
        fprintf(stderr, "Audio output for %s already exists\n", client->usrname);
    } else {
        if ((client->fp = popen("play -t s16 -c 1 -r 44100 -q --buffer 1024 -", "w")) == NULL) {
            fprintf(stderr, "Failed to open audio output for %s\n", client->usrname);
        }
    }
    return client;
}

int client_free(int sd) {
    ClientSettings *client = NULL;
    int i;
    for (i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i] != NULL && clients[i]->id == sd) {
            client = clients[i];
            break;
        }
    }
    if (client == NULL) {
        return -1;
    }
    if (client->fp != NULL) {
        pclose(client->fp);
        client->fp = NULL;
    }
    free(client);
    clients[i] = NULL;
}

ClientSettings *client_get(int sd) {
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (clients[i] != NULL && clients[i]->id == sd) {
            return clients[i];
        }
    }
    return NULL;
}

int recv_all(int socket, char *buffer, int length) {
    int total_received = 0;
    int bytes_left = length;
    int n;

    while (total_received < length) {
        n = recv(socket, buffer + total_received, bytes_left, 0);
        if (n <= 0) {
            // Handle error or disconnection
            return n;
        }
        total_received += n;
        bytes_left -= n;
    }

    return total_received;
}

int buf_w_bytes(char *sendbuf, int *len, int identifier, char *buf) {
    if (*len > BUF_READ) {
        fprintf(stderr, "%d bytes exceeds buffer size\n", *len);
        return -1;
    }
    snprintf(sendbuf, BUF_SEND, "%d.%04d>>",
```

```
    identifier, *len);
    bcopy(buf, sendbuf + 8, *len);
    *len += 8;
    return 0;
}

int buf_w_str(char *sendbuf, int *len, int identifier, char *str) {
    if (strlen(str) + 1 > BUF_READ) {
        fprintf(stderr, "String %s exceeds buffer size\n", str);
        return -1;
    }
    snprintf(sendbuf, BUF_SEND, "%d.%04ld>>%s", identifier, strlen(str) + 1, str);
    *len = 8 + strlen(str) + 1;
    return 0;
}

void handle_sigint(int sig) {
    for (int i = 0; i < MAX_CLIENTS; i++) {
        client_free(i);
    }
    close(s);
    exit(0);
}
```

multi_client.c

参考文献

[1] Bagwell, C. (n.d.). sox(1) - Linux man page. die.net. Retrieved June 29, 2024, from https://linux.die.net/man/1/sox

[2] GeeksforGeeks. (2023, December 13). Tail command in Linux with examples. GeeksforGeeks. https://www.geeksforgeeks.org/tail-command-linux-examples/

[3] pclose(3) - Linux man page. (n.d.). die.net. Retrieved June 30, 2024, from https://linux.die.net/man/3/pclose

[4] GeeksforGeeks. (2023, September 9). Named Pipe or FIFO with example C program. GeeksforGeeks. https://www.geeksforgeeks.org/named-pipe-fifo-example-c-program/