ネットワークプログラミング
最終レポート

222C1021 今村優希

2024 年 8 月 5 日

演習 2 WS による簡易チャット/LINE の作成

作成したものは chat.html，chat.js，chat.css で，server.js も多少変更を行った．chat.html(ソースコード 1) に関しては名前とメッセージを入力できるようした．そして，chat.js(ソースコード 2) によってその名前とメッセージの入力を確認し，ウェブソケットで送信し，受信した内容を表示をすることができる．chat.css(ソースコード 3) は，チャットのように表示させたかったため使用していたが，表示は中途半端になってしまった．

server.js(ソースコード 4) に関しては，受信・送信するデータが違ったため，変更を加えた．

上記の html を実行した結果が図 1 で，server.js の動作結果が図 2 である．html の画面において，先に左側で名前とメッセージを入力し，送信をした．次に，右側から送信を行った．コンソールには 2 つのメッセージの内容が表示されているのが確認でき，また自分が入力したものは緑色で表示され，相手が入力してきたものは白色で表示されるよう CSS を編集している．

ソースコード 1　chat.html

```
<!--222C1021 今村優希-->
<!--chat.html-->
<!DOCTYPE html><html lang="ja">
    <head>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="chat.css">
        <title> Chatアプリ </title>
    </head>
    <body>
        メッセージを送信して，受信を行う．
        <section class="content">

        </section>

        <send class="send">
            <form name="input2" action="">
                <input type="text" name="user" value="">
                <input type="text" name="message" value="">
                <input type="button" name="send" id="message_button" value="
                    送信">
            </form>
        </send>



        <div id="chat_area"></div>

        <!--json の埋め込み/実行-->
        <script src="chat.js">
         </script>
<!-- ※ 青字：JavaScriptイベントハンドラ 緑字：画面表示/メッセージ送信の
   WebSocket用メソッド -->
```

```
</body></html>
```

ソースコード 2　chat.js

```
//<!-- WebSocketのクライアントの生成 （※ localhost ～ マシン内で確認） -->
let ws = new WebSocket('ws://localhost:8080');

//<!-- 接続時に実行 ～ openイベントが発生 -->
ws.addEventListener('open', e => {
    console.log('WebSocket で接続');
});

//<!-- サーバからデータ受信 ～ messageイベント発生、e.
   dataとして格納データをコンソール表示 -->
ws.addEventListener('message', e => {
    //console.log(e.target);
    const receivedata = JSON.parse(e.data);

    console.log(receivedata);

    displayUser(receivedata.user, receivedata.from);
    displayMessage(receivedata.message, receivedata.from);
});


//wsを使って送信
document.getElementById('message_button').addEventListener('click', e =>{
    var user = document.input2.user.value;
    var message = document.input2.message.value;
    if(user && message){
        const data = JSON.stringify({user: user, message: message});
        ws.send(data);
        document.input2.message.value = '';
    }
});

//ユーザー表示
function displayUser(user, type){
    let ChatArea = document.getElementById('chat_area');
    let MessageElement = document.createElement('div');
    if(type == 'myself'){
        MessageElement.textContent = 'あなた';
        MessageElement.classList.add('user_myself');
        ChatArea.appendChild(MessageElement);
    } else{
```

```
        MessageElement.textContent = user;
        MessageElement.classList.add('user_other');
        ChatArea.appendChild(MessageElement);
    }
}

//メッセージを表示
function displayMessage(message, type){
    let ChatArea = document.getElementById('chat_area');
    let MessageElement = document.createElement('div');
    if(type == 'myself'){
        MessageElement.classList.add('message_myself');
        MessageElement.textContent = message;
        ChatArea.appendChild(MessageElement);
    } else {
        MessageElement.classList.add('message_other');
        MessageElement.textContent = message;
        ChatArea.appendChild(MessageElement);
    }
}
```

ソースコード 3　chat.css

```
@charset "utf-8";

.send{
    position: fixed;
    z-index: 1;
    bottom: 0;
    width: 100%;
    height: 50px;

    display: flex;
    justify-content: center;
    align-items: center;

    background-color: #ffffff;
}

.user_myself{
    text-align: right;
    padding: 5px;
    margin: 5px 0;
    border: 1px solid #111111;
    border-radius: 5px;
```

```
    background-color: #14e50c;
}

.user_other{
    text-align: left;
    padding: 5px;
    margin: 5px 0;
    border: 1px solid #111;
    border-radius: 5px;
    background-color: #f9f9f9;
}

.message_myself{
    text-align: center;
    padding: 5px;
    margin: 5px 0;
    border: 1px solid #111;
    border-radius: 5px;
    background-color: #14e50c;
}

.message_other{
    text-align: center;
    padding: 5px;
    margin: 5px 0;
    border: 1px solid #111;
    border-radius: 5px;
    background-color: #d7e3e9;
}
```

ソースコード 4　server.js

```
let ws = require('ws')                          //
    JSONのWS（WebSocket）パッケージの読み込み
var server = new ws.Server({port:8080});    // WebSocketサーバの生成 ～ ポー
    ト番号8080を利用
server.on('connection', ws => {             // WSサーバ機能の実行 ～ クライ
    アント接続時に対応
    ws.on('message', message => {           // クライアントからのデータ受信
        （messageに格納）時に実行
    console.log(message.toString("utf8"));                  // 1) サーバの
       標準出力にmessageの内容表示
    const GetMessage = JSON.parse(message);
    const user = GetMessage.user;
    const getmessage = GetMessage.message;
```

```
    const SendMessage = JSON.stringify({user: user, message: getmessage,
       from: 'myself'});
    const GetOtherMessage = JSON.stringify({user: user, message: getmessage,
        from: 'other'});

    ws.send(SendMessage.toString("utf8"));                    // 2) 送信
       クライアントのみにmessageの内容返信
    server.clients.forEach(client => {      // サーバ接続の全クライアントに
       対する処理の規定
        if (client !== ws)                  // 送信クライアントを除外する条
           件設定
            client.send(GetOtherMessage.toString("utf8"));          //
               3) 接続クライアントへのmessageの内容送信
            // ※ 表示で文字化けする場合 (message) の部分を全て (message.
               toString("utf8") ) に変更
        });
    });
    ws.on('close', () => {                  // 切断時に実行 (例：ページのリ
       ロード)
        console.log('close');                   // サーバの標準出力に close
           と表示
    });
}); // 緑字：画面表示/メッセージ送信のWebSocket用メソッド
```

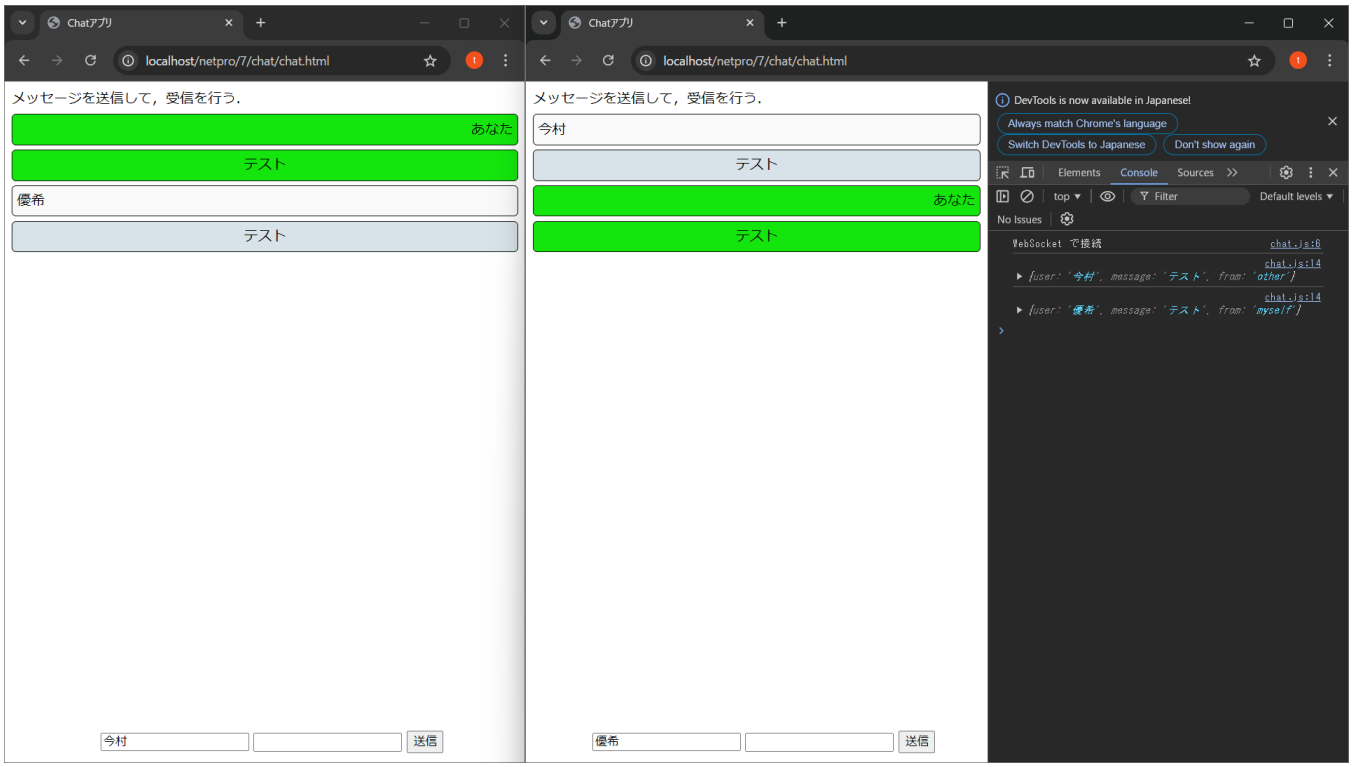

図 1　chat.html に接続し，メッセージの送信をした結果

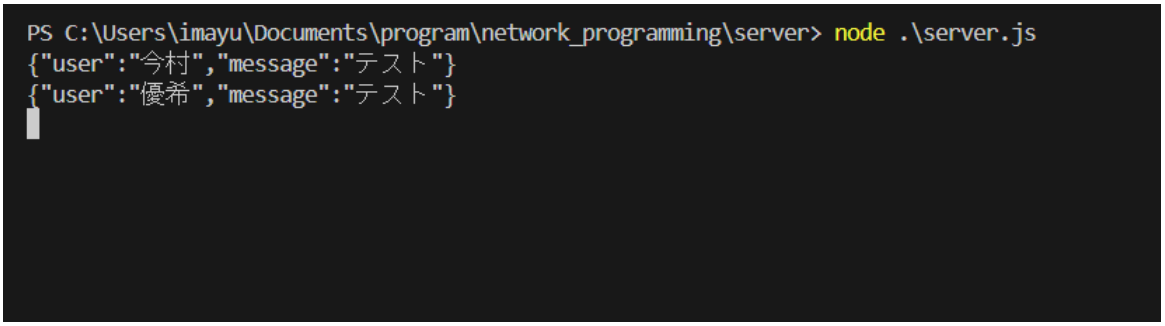

図 2　server.js の動作結果

演習 3 ソケット通信を利用した簡易 FTP の実現

3.1 ファイルを受信するクライアントプログラムの作成

ソースコード 5 は，ソケット通信を利用し，学生番号を入力し，受診した文字列をテキストファイルに格納するプログラムである．図 3 が，このプログラムを実行した際の出力である．

IP アドレス 131.206.37.50 に接続できた．222C1021.txt の内容を受診し，222C1021-copy.txt に書き込むことができた．

ソースコード 5　ftp.c

```
/*222C1021 今村優希*/
/*ftp.c*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <unistd.h>


int main(){
    int sockfd, fd;
    struct sockaddr_in address;
    char buf[80] = "\0";
    char end[10] = "exit_send";
    //INETドメイン、ストリームソケットを利用
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    //サーバー情報を設定
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("131.206.37.50");
    address.sin_port = htons(50000);

    //クライアントからの接続要求とサーバ情報を確認
    //川原研のサーバーに接続
    int res = connect(sockfd, (struct sockaddr *)&address, sizeof(address));

    if(res == -1){
        perror("error\n");
        exit(1);
```

```
    }
    printf("\n * server IP: %s, port: %d\n", inet_ntoa(address.sin_addr),
        ntohs(address.sin_port));


    //サーバーに学生番号を送信
    printf("Your ID:");
    scanf("%s", buf);
    write(sockfd, buf, strlen(buf));

    printf("read start\n");

    /*222C1021-copy.txtのオープン*/
    fd = open("222C1021-copy.txt", O_WRONLY);
    if(fd == -1){
        fprintf(stderr, "can't open the file\n");
        exit(1);
    }

    /*サーバからのデータ受信*/
    //入力がexit_sendになるまで読み込み、.txtに書き込む
    while(1){
        memset(buf, '\0', sizeof(buf)); // buf[]読み込み前に初期化
        int point = read(sockfd, buf, sizeof(buf));
        write(fd, buf, strlen(buf));
        if(strcmp(buf, end) == 0){
            printf("read finish\n");
            break;
        }
    }

    close(fd);      //222C1021-copy.txtのクローズ
    close(sockfd);  //ソケットの除去

}
```

```
imamura@imamura-VirtualBox:~/ドキュメント/network prog/7/ftp$ gcc -o ftp ftp.c
imamura@imamura-VirtualBox:~/ドキュメント/network prog/7/ftp$ ./ftp

 * server IP: 131.206.37.50, port: 50000
Your ID:222C1021
read start
read finish
```

図 3　`ftp.c` を実行した結果

3.2 取得ファイルの内容の確認，調査/報告

取得したテキストファイルは以下の内容であった．

> 課題：ルータにおける IP データグラム（パケット）転送
> 1. インターネットはパケット単位の蓄積交換方式（パケット交換方式）を採用するが、その利点と問題点を説明せよ。
> 2. 異種ネットワークを相互接続し、1. の主体となる「ルータ」において、到着パケットの転送先を決定する経路制御方式を一般的に何と呼ぶか答え、IPv4 アドレスを仮定した場合、どのように実現するか調査、説明せよ。
> 3. 代表的な経路情報交換プロトコルに RIP(Routing Information Protocol) と OSPF(Open Shortest Path First) がある。両者の違いを説明せよ。

課題 1

まず，蓄積交換方式とは，参考文献 [1] によると「インターネットではルータにおいてパケットをいったん蓄積して (store)，次のルータに転送する (forward) を意味している」とある．

したがって，蓄積交換方式のメリットとして，複数人で利用する際にインターネット資源を共有することができるという点が挙げられる．送信したいデータをパケットに分割することで，長いデータを送信したい等で資源を専有されることがなく，公平に使用することが可能である．

一方，デメリットとしてパケット到着の遅延と廃棄が発生する可能性があるという点が挙げられる．通信回線の利用率が高いとき，複数のパケットがルータに到着する可能性があり，バッファで待機することで遅延が発生する．さらに，この遅延が行き過ぎると，バッファにパケットが入らなくなり，パケットの廃棄が発生する．そうなるとパケットの再送が必要で効率が落ちる可能性がある．

課題 2

ルータにおける到着パケットの転送先を決定する経路制御方式を**経路制御**または**ルーティング**という．さらに，**ホップバイホップ制御**を使っている．その制御の中でも 2 種類の制御方法があり，**静的経路制御**と**動的経路制御**が存在する．静的経路制御は，ルータの管理者が経路表を自ら作成しているのに対して，動的経路制御は経路表をルータ自身で行っている．よって，現在では後者の動的経路制御で行っているのが一般的である．

IPv4 アドレスと仮定し動的経路制御を用いて転送を行うと考える．まず，ルータは経路表を作成し，各宛先ネットワークの経路情報を保存している．下記で詳しく解答する動的経路制御プロトコルを利用してルータ同士で経路情報を交換し，更新を行っている．そして，ルータがパケットを受診したときに宛先 IP アドレスを確認する．その IP ア

ドレスと経路表を参照して次の転送先を決定し，転送を行っている．

課題 3

RIP は，ルータが自分のルーティングテーブルを隣接するルータと共有し合うことで経路情報を更新する距離ベクトル型経路制御プロトコルである．各ルータは、宛先アドレスとその宛先に到達するためのメトリック (ホップ数) を交換している．ホップ数とは、パケットが宛先に到達するまでに通過するルータの数を指している．そのメトリックは定期的に隣接ルータに送信しており，その操作に時間がかかることがある．受け取ったメトリックから，Bellman-Ford アルゴリズムを用いて経路表を作成し，最短経路で宛先アドレスに到達できるようにしている．また，ホップ数の制限があるため，大規模なネットワークに向いておらず，小規模なネットワークで使用される．

OSPF はネットワーク内のルータが自分の接続 (リンク) の状態に関する情報を交換し，ネットワークの最適な経路を計算するリンク状態型経路制御プロトコルである．このプロトコルでは，ネットワーク内のすべてのルータおよびリンクに関する情報を収集し，それをもとに LSDB というリンク状態情報データベースを作成する．このデータベースは帯域幅や遅延，その他のメトリックを使用しており，ホップ数のみを使用する RIP よりも正確なネットワークのマップを作成することが可能である．そして，リンクの状態が変化したときのみアップデートを行うため，ネットワークのトラヒックが減る．このような方法から，OSPF は大規模なネットワークで使用されており，今日のルータの多くで採用されている．

3.3 3.1 の機能を提供するサーバープログラムの作成

作成したものは，ftp-server.c と ftp-client.c の 2 種類であり，ソースコード 6,7 に示している．また，これらのプログラムを実行した結果が図 4 である．サーバー側ではクライアントとと送受信を行う際に子プロセスを作成する．クライアント側で学生番号を入力すると，サーバ側で学生番号を受け取り，学生番号.txt をオープンする．サーバー側でも `exit_send` が `buf` の中にあると読み込み，送信を終了する．クライアントの動きは 3.1 と同じである．図 4 の状況からわかる通り，クライアントが終了したとしてもサーバーは受診待ちの状態である．

また，図 5 は送信する内容 (`222C1021.txt`) で，図 6 は受診した内容 (`222C1021-test.c`) である．

ソースコード 6　ftp-server.c

```
/*222C1021 今村優希*/
/*ftp-server.c*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <unistd.h>



int main(){
    int sockfd_s, fd, pid, pwait;
    struct sockaddr_in address_s;
```

```
char buf[80] = "\0";
char txt[] = ".txt";
char end[10] = "exit_send";

//INETドメイン、ストリームソケットを利用
sockfd_s = socket(AF_INET, SOCK_STREAM, 0);

//サーバソケット=(IP: 10.0.2.15, ポート:5000)に設定
address_s.sin_family = AF_INET;
address_s.sin_addr.s_addr = inet_addr("10.0.2.15");
address_s.sin_port = htons(5000);

bind(sockfd_s, (struct sockaddr *)&address_s, sizeof(address_s));

//要求受付の準備
listen(sockfd_s, 5);
printf("sertver waits\n");

//接続要求の許可と情報確認
struct sockaddr_in address_c;
unsigned int length_c = sizeof(address_c);

/*子プロセスでファイルの送受信を行う*/
pid = fork();
while(1){
    int sockfd_c = accept(sockfd_s, (struct sockaddr *)& address_c, &
       length_c);
    pid = fork();
    if(pid == 0){
        printf("\n * request from client IP: %s, port %d\n", inet_ntoa(
           address_c.sin_addr), ntohs(address_c.sin_port));
        //学生番号の取得
        memset(buf, '\0', sizeof(buf));
        read(sockfd_c, buf, sizeof(buf));

        //学生番号と.txtを結合
        strcat(buf, txt);
        printf("%s\n", buf);    //debug
        fd = open(buf, O_RDONLY);
        if(fd == -1){
            printf("file can't the open\n");
            exit(1);
        }
```

```
            printf("start write\n");
            while(1){
                memset(buf, '\0', sizeof(buf)); // buf[]読み込み前に初期化
                read(fd, buf, sizeof(buf));     //ファイルの内容をbufに格納
                write(sockfd_c, buf, strlen(buf));//送信
                if(strstr(buf, end) != NULL){
                    printf("finished write\n");
                    break;
                }
            }
            close(sockfd_c);
            close(sockfd_s);

            exit(1);
        } else{
            int status;
            pwait = waitpid(-1, &status, WNOHANG);
            printf("sarver wait\n");
        }
    }
 }
```

ソースコード 7　ftp-client.c

```
/*222C1021 今村優希*/
/*ftp-client.c*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <unistd.h>


int main(){
    int sockfd, fd;
    struct sockaddr_in address;
    char buf[80] = "\0";
    char end[10] = "exit_send";
    //INETドメイン、ストリームソケットを利用
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

```
//サーバー情報を設定
address.sin_family = AF_INET;
address.sin_addr.s_addr = inet_addr("10.0.2.15");
address.sin_port = htons(5000);

//クライアントからの接続要求とサーバ情報を確認
//サーバーに接続
int res = connect(sockfd, (struct sockaddr *)&address, sizeof(address));

if(res == -1){
    perror("error\n");
    exit(1);
}
printf("\n * server IP: %s, port: %d\n", inet_ntoa(address.sin_addr),
   ntohs(address.sin_port));

//サーバーに学生番号を送信
printf("Your ID:");
scanf("%s", buf);
write(sockfd, buf, strlen(buf));

printf("read start\n");

/*222C1021-test.txtのオープン*/
fd = open("222C1021-test.txt", O_WRONLY);
if(fd == -1){
    fprintf(stderr, "can't open the file\n");
    exit(1);
}

/*サーバからのデータ受信*/
//入力がexit_sendになるまで読み込み、.txtに書き込む
while(1){
    memset(buf, '\0', sizeof(buf)); // buf[]読み込み前に初期化
    read(sockfd, buf, sizeof(buf));
    write(fd, buf, strlen(buf));
    if(strstr(buf, end) != NULL){
        printf("read finish\n");
        break;
    }
}

close(fd);      //222C1021-test.txtのクローズ
```

```
    close(sockfd);  //ソケットの除去

}
```

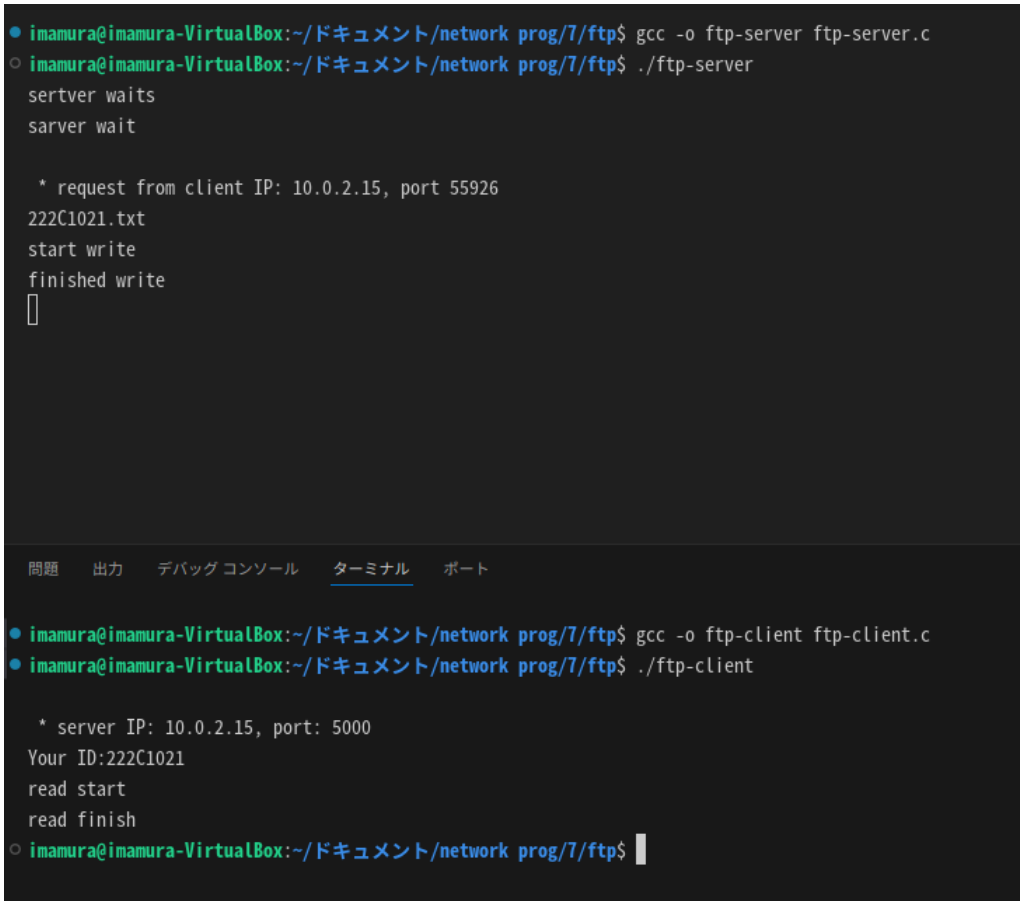

図 4　ftp-server.c と ftp-client.c の実行結果

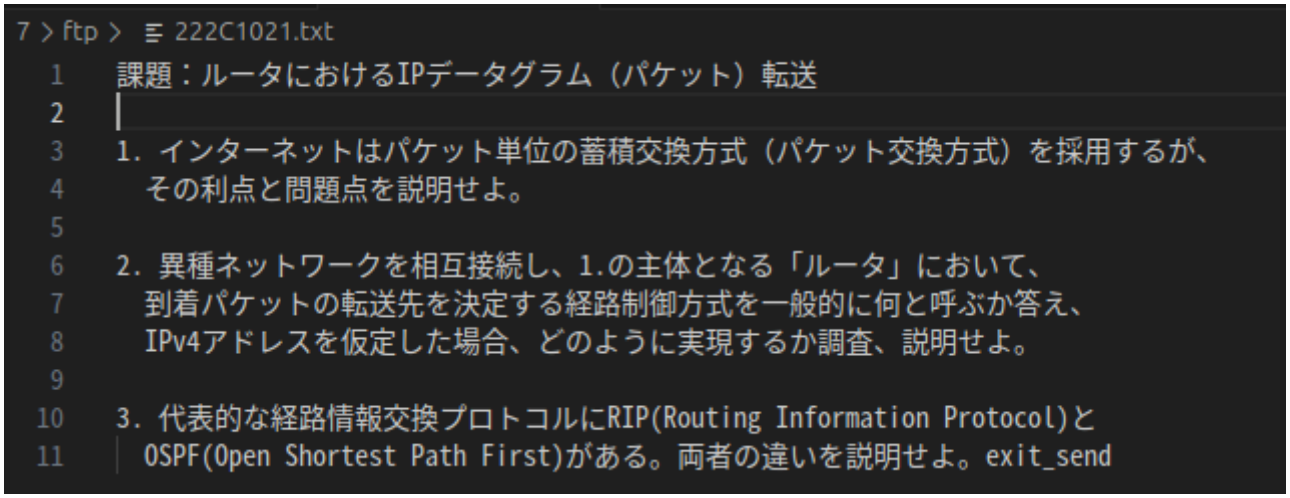

図 5　送信元のデータ (222C1021.txt)

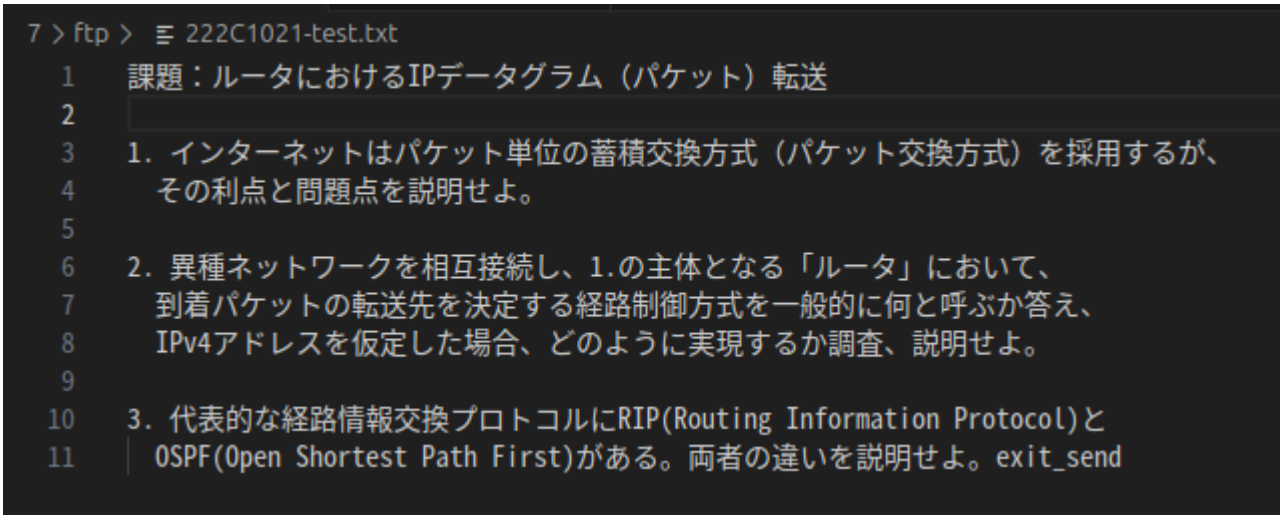

図 6　受診したデータ (222C1021-test.c)

演習 4 サーバー機能のクラウドへのデプロイ

演習 3 で作成した `ftp-server.c` を Google Cloud にデプロイし，クライアントから接続の挑戦を行った．結果としては `ftp-server.c` に接続できず，理想のことができなかったが，クラウドの使い方がある程度理解でき，かなり複雑なんだなと言うことが体験できた．

以下では自分が行った内容と結果を記述していく．

1. アカウントの作成

Google Cloud を始めるにあたり，アカウントの作成を行った．

2. Google Cloud Platform にログイン，プロジェクトを作成する

まずは，Google Cloud Platform にログインした．その後，プロジェクトの作成を行い，そのプロジェクトにログインする．そのプロジェクトのホーム画面を図 7 に表示している．

図 7　プロジェクトのホーム画面

3. VM の作成

`ftp-server.c` を実行するための Ubuntu を作成した．まずは，Google Cloud のログイン画面から `Compute Engine` という部分というところに移動し，API を有効かした．その後インスタンスを作成し，実際に VM の作成に取り掛かった．インスタンスを作成した結果が図 8 である．`ftp-server-20240805` というインスタンスが作成されている．

図 8　インスタンスの作成結果

4. `ftp-server.c` 等のファイルをアップロードする

SSH で先程作成したインスタンスに接続し，`ftp-server.c` と `222C1021.txt` をアップロードした．SSH 接続とアップロードの結果を図 9 に示している．

`ftp-server.c` に関して，IP アドレスとポート番号を変更する必要があったので変更を加えた．`ftp-client.c` も IP アドレスとポート番号の変更を行った．

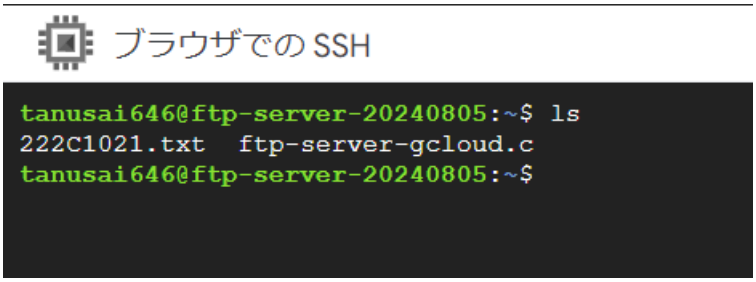

図 9　ファイルのアップロード状況

5. ファイアーウォールの設定

先ほど設定したポート番号で接続できるようにファイアーウォールの設定を行った．作成した結果が図 10 である．今回設定したものは `for-ftp-server` であり，すべての IP アドレスに対して tcp:50000 が許可されている．

図 10　設定したファイアーウォールの状況

6. クライアントから接続

最終的に IP アドレスとポート番号を変更した `ftp-server-gcloud.c` を Cloud の VM 上で実行し，`ftp-client-gcloud.c` を自分のパソコンで実行を行った．その結果が図 11，12 である．`ftp-server-gcloud.c` の実行は問題なく行われている．しかし，`ftp-client-gcloud.c` では `Conection Refuced` というエラーが発生し接続ができなかった．

図 11　ftp-server-gcloud.c の実行結果

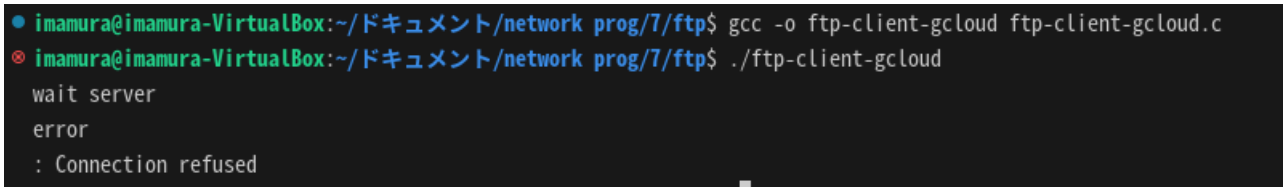

図 12　ftp-client-gcloud.c の実行結果

ファイアーウォールで設定したルールが適用された回数を意味する**ヒットカウント**が増加していることから，クライアントからの接続要求が立っていることは確認できた．また，そのログにて接続した痕跡が見られた (ソースコード 8)．ただ，これらの情報に関して調べても接続できない理由ははっきりしなかったため，ここであきらめた．

ソースコード 8　実行したときのファイアーウォールのログ

```
{
  "insertId": "1eiw5avefena0",
  "jsonPayload": {
    "rule_details": {
      "priority": 1000,
      "ip_port_info": [
        {
          "ip_protocol": "TCP",
          "port_range": [
            "50000"
          ]
        }
      ],
      "direction": "INGRESS",
      "source_range": [
        "0.0.0.0/0"
      ],
      "reference": "network:default/firewall:for-ftp-server",
      "action": "ALLOW"
    },
    "remote_location": {
      "region": "Fukuoka",
```

```
      "country": "jpn",
      "continent": "Asia",
      "city": "Iizuka"
    },
    "disposition": "ALLOWED",
    "instance": {
      "zone": "asia-northeast1-a",
      "vm_name": "ftp-server-20240805",
      "region": "asia-northeast1",
      "project_id": "ftp-server-431407"
    },
    "vpc": {
      "vpc_name": "default",
      "project_id": "ftp-server-431407",
      "subnetwork_name": "default"
    },
    "connection": {
      "dest_ip": "10.146.0.3",
      "protocol": 6,
      "dest_port": 50000,
      "src_ip": "131.206.231.102",
      "src_port": 58006
    }
  },
  "resource": {
    "type": "gce_subnetwork",
    "labels": {
      "location": "asia-northeast1",
      "subnetwork_id": "1817687114020573949",
      "project_id": "ftp-server-431407",
      "subnetwork_name": "default"
    }
  },
  "timestamp": "2024-08-05T04:15:03.533411117Z",
  "logName": "projects/ftp-server-431407/logs/compute.googleapis.com%2
     Ffirewall",
  "receiveTimestamp": "2024-08-05T04:15:07.395005699Z"
}
```

備考

https://github.com/tanusai646/network_program/tree/main/7

参考文献

[1] 尾家祐二 後藤滋樹 小西和憲 西尾章治郎, 1 インターネット入門, 株式会社岩波書店, 2002
[2] 堀良影 池永全志 門林雄基 後藤滋樹, 2 ネットワーク相互接続, 株式会社岩波書店, 2001