

学籍番号: 09B20033

城間大幹

2022 年 8 月 24 日

1 課題 4-1

1.1 実装方針とアルゴリズム

1.1.1 クライアントプログラム

指導書で指定された状態 c1 から c6 を作る形で実装した.

図??に作成したクライアントプログラムのフローチャートを示す.

次に各状態について、該当箇所のコードを参照しながら説明する.

- c1(初期状態)

該当するソースコードを以下に示す.

```
int sock;
int c2=0,c3=0,c4=0,c5=0;
int result=0;
char buf[1024];
struct sockaddr_in host;
struct hostent *hp;
fd_set rfds;
struct timeval tv;

if (argc != 3) {
    fprintf(stderr,"Usage: %s hostname message\n",argv[0]);
    exit(1); }

if ( ( sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP) ) < 0 ) {
    perror("client: socket");
    exit(1); }

bzero(&host,sizeof(host));
host.sin_family=AF_INET;
host.sin_port=htons(10140);

if ( ( hp = gethostbyname(argv[1]) ) == NULL ) {
    fprintf(stderr,"unknown host %s\n",argv[1]);
    exit(1);
}
bcopy(hp->h_addr,&host.sin_addr, hp->h_length);
result=connect(sock,(struct sockaddr *)&host,sizeof(host));
if ((result)==-1){
    perror("cannot \n");
    exit(1);
}
else{
    printf("connected\n");
    c2=1;
}
```

ホストとの接続方法は課題 2,3 で用いた方法と同じであるためここでは説明を割愛する。connect 関数を実行し、接続が確立すれば状態 c2 に入るためのフラグ、c2 を 1 として、次の状態に移る。

- c2 (参加)

該当するソースコードを以下に示す.

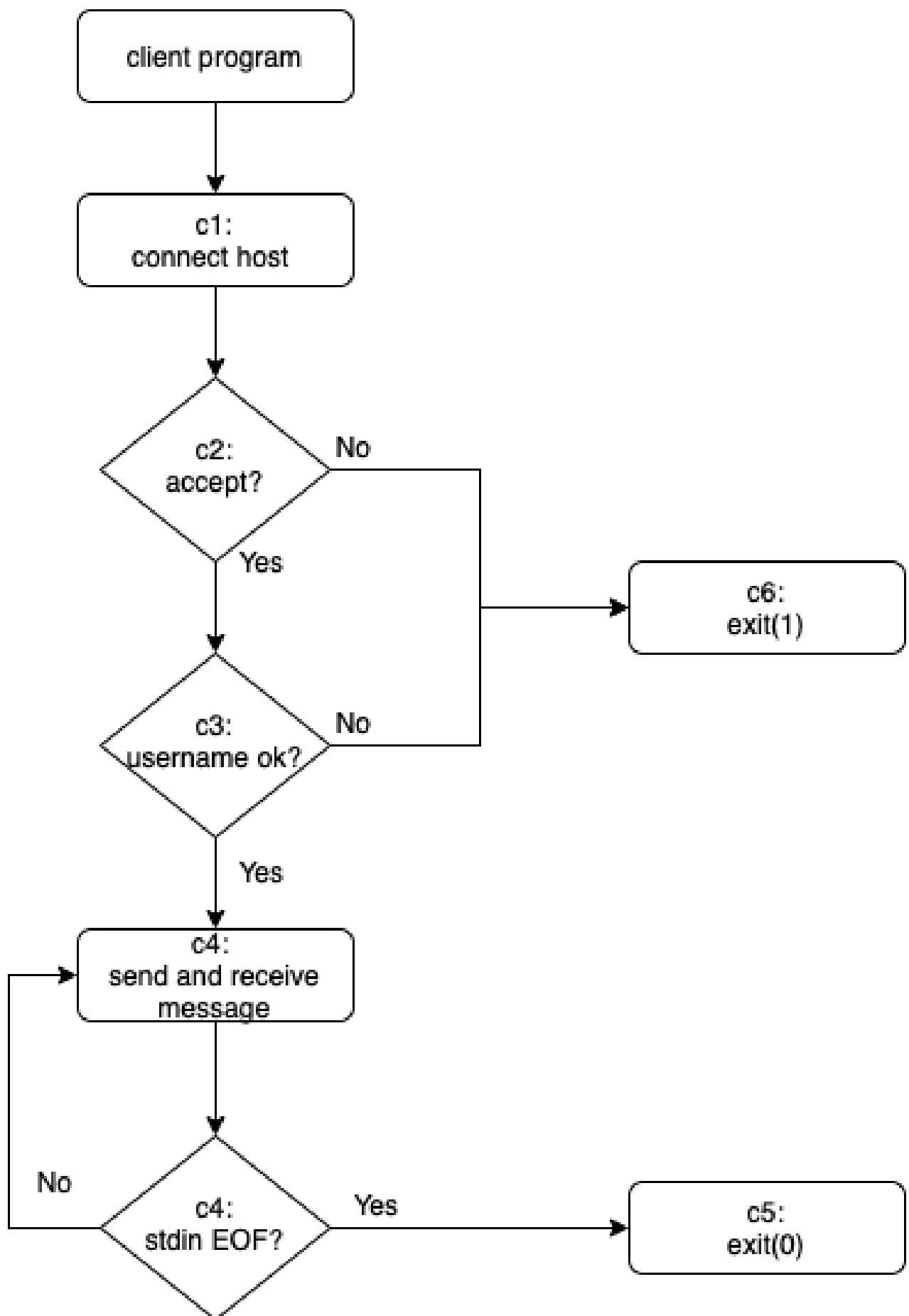


図1 作成したクライアントプログラムのフローチャート

```

if(c2==1){
    //c2
    memset(buf, '\0', sizeof(buf));
    if ( ( read(sock,buf,sizeof(buf)) ) < 0) {
        perror("read");
    }
    else {
        printf("receive: %s\n",buf);

        if( !strcmp( "REQUEST ACCEPTED\n", buf, strlen("REQUEST ACCEPTED\n"))){
            c3=1;
        }
        // もしメッセージが接続受理 ("REQUEST ACCEPTED\n") ならば状態 c3 へ
        else c6_exception(sock);
        //さもなければ (接続拒否または何らかの例外) 状態 c6 へ移る.
    }
}

```

サーバーからメッセージを read 関数により受信し、そのメッセージが REQUEST ACCEPTED\n であるかを strcmp 関数により判別する。接続受理のメッセージであれば c3 に遷移するためのフラグ、c3 を 1 とし、そうでなければ例外処理を行う関数 c6_exception を実行する。なお関数 c6_exception については状態 c6 で詳細を説明する。

- c3(ユーザー名登録)

該当するソースコードを以下に示す。

```

if (c3==1){
    if(write(sock, argv[2], /*sizeof*/strlen(argv[2]))==-1){
        perror("writeforsend\n");
        exit(1);
    }

    memset(buf, '\0', sizeof(buf));

    if ( ( read(sock,buf,sizeof(buf)) ) < 0) {
        perror("read");
    }
    else {
        printf("receive: %s\n",buf);

        if( !strcmp( "USERNAME REGISTERED\n", buf, strlen("REQUEST ACCEPTED\n"))){
            c4=1;
        }
        // もしメッセージが接続受理 ("USERNAME REGISTERED\n") ならば状態 c4 へ
        else c6_exception(sock);
        //さもなければ (接続拒否または何らかの例外) 状態 c6 へ移る.
    }
}

```

引数で指定されたユーザー名を write 関数によりサーバーに送信する。その後サーバーからメッセージを read 関数により受信し、そのメッセージが USERNAME REGISTERED\n であるかを strcmp 関数により判別する。ユーザー名登録完了のメッセージであれば c4 に遷移するためのフラグ、c4 を 1 とし、そうでなければ例外処理を行う関数 c6_exception を実行する。なお関数 c6_exception については状態 c6 で詳細を説明する。

- c4 (メッセージ送受信)

該当するソースコードを以下に示す.

```
if (c4==1){
    do {

        if (finish==1){
            c4=0;
            c5=1;
            break;
        }

        FD_ZERO(&rfds);
        FD_SET(0,&rfds);
        FD_SET(sock,&rfds);
        tv.tv_sec = 1;
        tv.tv_usec = 0;
        memset(buf, '\0', sizeof(buf));
        if(select(sock+1,&rfds,NULL,NULL,&tv)>0 ){

            if(FD_ISSET(sock,&rfds)){
                size_t readlen=read(sock,buf,sizeof(buf));
                if (readlen < 0) {
                    perror("read");
                    exit(1);
                }
                else if (readlen == 0) {
                    c5=1;
                    break;
                }
                else {
                    size_t length=strlen(buf);
                    if (length > 0 && buf[length - 1] == '\n') {
                        buf[--length] = '\0';
                    }
                    printf("%s",buf);
                    printf("\n");
                }
            }
        }

        if(FD_ISSET(0,&rfds)) {
            myalarm(TIMEOUT);

            if (fgets(buf, sizeof(buf), stdin) == NULL) {
                // 標準入力が EOF なら状態 c5 へ
                c5=1;
                break;
            }
        }

        else{
            if(write(sock, buf,strlen(buf))==-1){
                perror("writeforclient\n");
                exit(1);
            }
        }
    }
}
```

```

        }
    }
} while (1);

```

メッセージの送受信方法は課題 2,3 で用いた方法と同じであるためここでは詳細な説明は割愛する。状態 c4 は while 文によりループしているため、常にメッセージの送受信を受け付けるようになっている。標準入力が Ctr+d などによる EOF であったとき、c5 に遷移するためのフラグ、c5 を 1 とし while 文中から break により離脱する。なお myalarm(TIMEOUT) などは課題 4-2 での追加機能であるためここでは説明を割愛する。

- c5(離脱)

該当するコードを以下に示す。

```

if(c5==1){
    printf("finished\n");
    close(sock);
    exit(0);
}

```

生成したソケットを閉じて、exit(0) によりプログラムを終了する。

- c6(例外処理)

該当するコードを以下に示す。

```

void c6_exception(int sock){
    perror("error");
    close(sock);
    exit(1);
}

```

生成したソケットを閉じて、exit(1) によりプログラムを終了する。

1.1.2 サーバプログラム

通常課題を発展課題で指定されていたスレッドを用いて作成した。すなわち作成したサーバープログラムは一つである。

図??に作成したサーバープログラムのフローチャートを示す。基本的には指導書に記載された状態遷移を参考にしている。

なお今回サーバープログラムを作成するにあたって、発展課題で指定されていたスレッドを用いたのでその流れを説明する。メインとなるプログラムでは新規の接続受付、ユーザー名の登録を行う。そしてクライアントの会話への参加が決定すれば、そのクライアント用にスレッドを生成し、そのスレッド内で発言を他のクライアントに転送したり、他のクライアントの発言を受信したりする操作を行う。したがって、新規の接続を監視と、クライアントごとのスレッドの最大で 6 個の独立した操作が行われることになる。

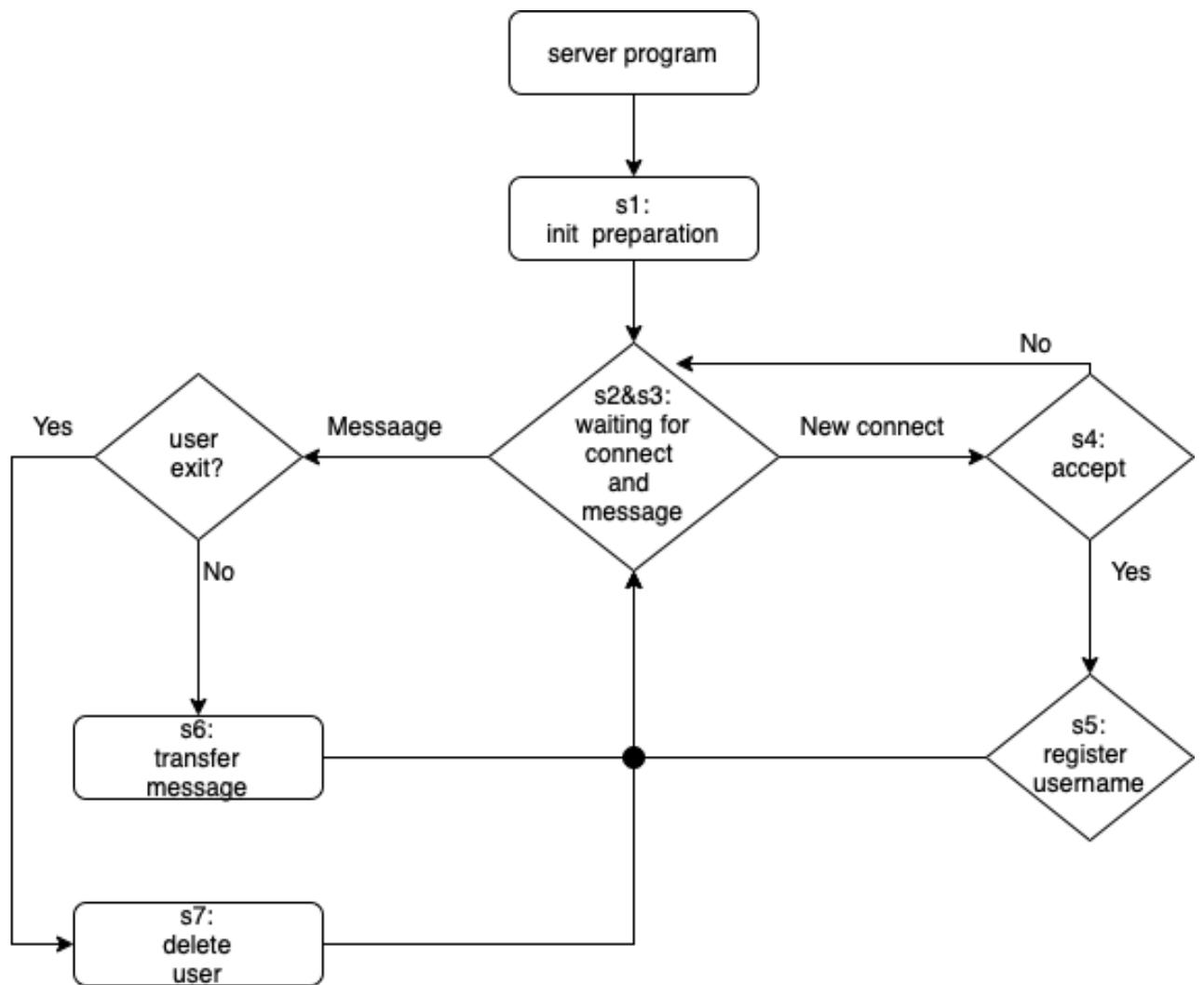


図2 作成したサーバープログラムのフローチャート

続いて主なグローバル変数の定義と用途について表??に示す.

表 1 グローバル変数の定義と用途

変数名	型	測定結果
csock[MAXCLIENTS+1]	int 型配列	csock[0] は新規接続用. その他は各クライアントのソケットを格納
rbuf[MAXCLIENTS+1][1024]	int 型二次元配列	rbuf[0] は新規接続用. その他は各クライアントのメッセージ操作で使用
user_name_list[MAXCLIENTS+1][1024]	int 型二次元配列	ユーザー名を登録する. +1 は記述を簡単にするためである.
k	int 型	現在接続中のクライアントの数を示す.
flag1 ~ flag5	int 型	5つのスレッドの使用状況を示す. 1 の時は使用中である.

スレッドではグローバルに定義した変数などをリアルタイムの共有情報として使用できるため、スレッドを生成するタイミングに関わらず、各スレッド内から現在参加中のクライアント数、クライアントのアドレスが参照でき、メッセージの送受信が容易に行える。また、flag はその共有資源である csock、rbuf、user_name_list などを管理するために使用している。

次に各状態について、コードを該当箇所のコードを参照しながら説明する。

- s1 (初期状態)

該当するコードを以下に示す。

```

int sock;
int temp;
int reuse;
struct sockaddr_in svr;
pthread_t handle1;
pthread_t handle2;
pthread_t handle3;
pthread_t handle4;
pthread_t handle5;
pthread_mutex_init(&mutex, NULL);

if ((sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))<0) {
    perror("socket");
    exit(1);
}

reuse=1;
if(setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&reuse,sizeof(reuse))<0) {
    perror("setsockopt");
    exit(1);
}

bzero(&svr,sizeof(svr));
svr.sin_family=AF_INET;
svr.sin_addr.s_addr=htonl(INADDR_ANY);
svr.sin_port=htons(10140);

if(bind(sock,(struct sockaddr *)&svr,sizeof(svr))<0) {

```

```

    perror("bind");
    exit(1);
}

if (listen(sock,5)<0) {
    perror("listen");
    exit(1);
}

```

クライアントとの接続方法は課題 2,3 で用いた方法と同じであるためここでは説明を割愛する。

なお変数についてだが、temp は後に実行する関数の戻り値を格納するために使用し、handle1 から handle5 は各クライアントごとに生成するスレッドの操作で用いるためのものである。詳細は次の状態で説明する。

- s2 (入力待ち) & s3 (入力処理)

該当するコードを以下に示す。

```

while(1){
    pthread_mutex_lock(&mutex);
    if(k>0){

        if (flag1!=1)  pthread_create(&handle1, NULL,client_socket, NULL);
        else if (flag2!=1) pthread_create(&handle2, NULL,client_socket, NULL);
        else if (flag3!=1) pthread_create(&handle3, NULL,client_socket, NULL);
        else if (flag4!=1) pthread_create(&handle4, NULL,client_socket, NULL);
        else if (flag5!=1) pthread_create(&handle5, NULL,client_socket, NULL);

    }
    pthread_mutex_unlock(&mutex);

    temp=connect_check(sock);

    if(temp!=-1) {
        pthread_mutex_lock(&mutex);
        k++;
        pthread_mutex_unlock(&mutex);
        for (int i=1;i<MAXCLIENTS+1;i++){
            if ( csock[i]==-1){
                csock[i]=temp;
                break;
            }
        }
    }
    else  close(csock[0]);
    for (int i=1;i<MAXCLIENTS+1;i++){
        printf("user_name: %s\n",user_name_list[i]);
        printf("addr: %d\n",csock[i]);
    }
}
}

pthread_mutex_destroy(&mutex);

```

if (k > 0) 内で各クライアントに対するスレッドを生成している。flag による判定で使用できるスレッドを判定しており、クライアントの参加、退出に応じて handle の番号が若いところからスレッドが生成されるようになっ

ている。スレッドで呼び出す関数 client_socket については s6, s7 で説明する。

そしてスレッドを呼び出す側のプログラム、main 関数内では関数 connect_check により新規の接続とユーザ名の処理を行なっている。connect_check の戻り値は新規クライアントのソケットになっており、それを csock が-1 になっている、すなわち未使用の配列 csock の要素として格納する。その後ループが戻って再度スレッドを生成する際に、参加を受け付けたクライアントのスレッドを生成する。なお関数 connect_check の詳しい内容については s4, s5 で説明する。

- s4 (参加受け付け) & s5 (ユーザ名登録)
該当するコードを以下に示す。

```
int connect_check(int sock){
    struct sockaddr_in clt;
    struct hostent *cp;
    unsigned int clen;
    int client_sock=-1;
    int i,flag=0;

    clen = sizeof(clt);
    if ( ( csock[0] = accept(sock,(struct sockaddr *)&clt,&clen) ) < 0 ) {
        perror("accept");
        exit(2);
    }

    pthread_mutex_lock(&mutex);
    if(k<MAXCLIENTS) {
        write(csock[0], "REQUEST ACCEPTED\n", strlen("REQUEST ACCEPTED\n"));

        cp = gethostbyaddr((char *)&clt.sin_addr,sizeof(struct in_addr),AF_INET);
        printf("[%s]\n",cp->h_name);

        memset(rbuf[0], '\0', sizeof(rbuf[0]));
        if ( ( read(csock[0],rbuf[0],sizeof(rbuf[0])) ) < 0 ) {
            perror("read");
            exit(1);
        }

        else{
            size_t length = strlen(rbuf[0]);
            if (length > 0 && rbuf[0][length - 1] == '\n') {
                rbuf[0][--length] = '\0';
            }
            for (i=1;i<MAXCLIENTS+1;i++){
                if(strcmp(rbuf[0],user_name_list[i])==0){
                    write(csock[0], "USERNAME REJECTED\n", strlen("USERNAME REJECTED\n"));
                    flag=1;
                    break;
                }
            }
            if (flag!=1){
                for (i=1;i<MAXCLIENTS+1;i++){
                    if (user_name_list[i][0]=='\0'){
                        write(csock[0], "USERNAME REGISTERED\n", strlen("USERNAME REGISTERED\n"));
                        memset(user_name_list[i], '\0', sizeof(user_name_list[i]));
                    }
                }
            }
        }
    }
}
```

```

        strcpy(user_name_list[i], rbuf[0]); //register username
        client_sock=csock[0]; //register socket
        break;
    }
}
}

}

else{
    write(csock[0], "REQUEST REJECTED\n", strlen("REQUEST REJECTED\n"));
}

pthread_mutex_unlock(&mutex);

return client_sock;
}

```

accept 関数により新規クライアントの接続を受け付け、ソケットを csock[0] に格納する。まず、もし現在の参加クライアント数 k が MAXCLIENTS 未満であれば、接続受理のメッセージ（文字列”REQUEST ACCEPTED\n”）を、さもなければ接続拒否のメッセージ（文字列”REQUEST REJECTED\n”）を送信し、-1 を返す。

接続を受理した場合、次にユーザー名の登録を行う。rbuf[0] にクライアントから送信されたユーザーネームを格納するが、この際改行文字が含まれているため改行文字を消去する操作を行う。ユーザーネームが既に登録されていないかの判定は関数 strcmp により行い、既に登録されていた場合は登録拒否のメッセージ（文字列”USERNAME REJECTED\n”）を返し、さもなければ登録完了メッセージ（文字列”USERNAME REGISTERED\n”）を送信する。

ユーザーネームを配列 user_name_list に登録する場合、1～MAXCLIENTS の配列内要素で空、かつ若い番号のところに格納する。0 番目を使用しなかったのは記述のしやすさのためで、特に理由はない。

そして新規クライアントの参加が決まればそのクライアントのソケットを戻り値として返す。何かしらの問題があれば-1 返すようになっている。

- s6 (メッセージ配信) & s7 (離脱処理)
該当するコードを以下に示す。

```

void *client_socket(void *arg) {
    int client_number;
    char temp[99];
    char yajirushi[]{"==>"};
    char fin_message[]{"connection ended\n"};
    if (flag1!=1) {
        flag1=1;
        client_number=1;
    }
    else if (flag2!=1) {
        flag2=1;
        client_number=2;
    }
    else if (flag3!=1) {
        flag3=1;
        client_number=3;
    }
}

```

```

    }

else if (flag4!=1) {
    flag4=1;
    client_number=4;
}

else {
    flag5=1;
    client_number=5;
}

do {
    fd_set rfds;
    struct timeval tv;
    FD_ZERO(&rfds);
    FD_SET(0,&rfds);
    FD_SET(csock[client_number] ,&rfds);
    tv.tv_sec = 1;
    tv.tv_usec = 0;
    if(select(csock[client_number]+1,&rfds,NULL,NULL,&tv)>0 ){ //s6

        memset(rbuf[client_number], '\0', sizeof(rbuf[client_number]));
        if(FD_ISSET(csock[client_number] ,&rfds)) {

            size_t readlen=read(csock[client_number] ,rbuf[client_number] ,sizeof(rbuf[client_number]));
            if ( readlen < 0) {
                perror("read");
                exit(1);
            }
            else if (readlen == 0) { //ctr+D
                //s7
                for (int j=1; j<=MAXCLIENTS; j++){
                    if(j!=client_number && csock[j]!=-1){
                        memset(temp, '\0', sizeof(temp));
                        strcat(temp,user_name_list[client_number]);
                        strcat(temp,yajirushi);
                        strcat(temp,fin_message);
                        write(csock[j],temp,sizeof(temp));
                    }
                }
            }
            printf("%s: exit\n",user_name_list[client_number]);
            close(csock[client_number]);

            memset(user_name_list[client_number], '\0', sizeof(user_name_list[client_number]));
            csock[client_number]=-1;
            pthread_mutex_lock(&mutex);
            k--;
            pthread_mutex_unlock(&mutex);

            switch (client_number){
                case 1:
                    flag1=0;
                    break;

```

```

        case 2:
            flag2=0;
            break;

        case 3:
            flag3=0;
            break;

        case 4:
            flag4=0;
            break;

        case 5:
            flag5=0;
            break;
    }
}

else {
    write(1,user_name_list[client_number],sizeof(user_name_list[client_number]));
    write(1,"==>",sizeof("==>"));
    write(1,rbuf[client_number],sizeof(rbuf[client_number]));
    //write で他の socket にも出力
    for (int j=1; j<=MAXCLIENTS; j++){
        if(j!=client_number && csock[j]!=-1){
            memset(temp, '\0', sizeof(temp));
            strcat(temp,user_name_list[client_number]);
            strcat(temp,yajirushi);
            strcat(temp,rbuf[client_number]);
            write(csock[j],temp,sizeof(temp));

        }
    }
}
}

}
while (1);
return NULL;
}

```

ここがスレッドで呼び出される箇所であり、各クライアントのメッセージの送受信を行う。まず flag1~5 による判定で client_number を決定する。これは共有情報である配列 csock, rbuf, user_name_list のどの要素を参照するかを決定するために行なっており、他のスレッドとの競合を防ぐためである。

次に select 関数によりクライアント間でのメッセージの送受信を行うが、送受信などに関する関数周りは課題 2,3 でのものと同様であるため詳細な説明は割愛し、複数のクライアントへのメッセージの送信と離脱処理について説明する。

まず複数のクライアントへのメッセージの送信について説明する。クライアントプログラムから送信されたメッセージを `rbuf[client_number]` に格納する。そして `rbuf[client_number]` を `write` 関数により、他のクライアントプログラムへ転送する。その方法は配列 `csock` を 1～`MAXCLIENTS` まで順番に参照し、-1 でないソケットに対して `write` 関数を実行すればよい。`csock` はグローバル変数であり、かつスレッドは資源を共有できるためス

レッド生成時の csock にはなかった情報、すなわち新規に参加したクライアントのソケットなども参照できるのである。

続いて離脱処理について説明する。まず離脱した旨を他のクライアントに送信する。この方法はメッセージの送信方法と同様である。次に client_number に対応する共有情報である配列 csock, user_name_list などの内容を変更する。csock[client_number] は close によりソケットを閉じた後-1 を代入し、user_name_list[client_number] は memset により内容をリセット、そして会話参加中のクライアントの数を示す k の数を減らす。また client_number に応じて flag の値を変更する。これにより退出したクライアントが使用していたスレッドを空け、再度使用できる状態にすることができる。

1.2 動作確認

- サンプルサーバと作成したクライアントプログラム間での動作結果
図??に動作結果を示す。

The figure shows three terminal windows on a Linux desktop. The top-left window (Terminal) shows a client session with user 'daiki'. The client sends 'Hello' and 'HI!', and receives responses from 'siroma'. The top-right window (Terminal) shows a server session where a long user name is rejected with the message 'Too long user name. The maximum length is 99. The overflowed part is not used.' The bottom-left window (Terminal) shows another client session with user 'siroma', which also has a similar interaction with 'daiki'.

```
d-siroma@exp001:~/Desktop/chat/client$ ./a.out exp008 daiki
connected
receive: REQUEST ACCEPTED
receive: USERNAME REGISTERED
daiki
daiki >daiki
siroma >siroma
Hello
daiki >Hello
siroma >HI!
^C
d-siroma@exp001:~/Desktop/chat/client$ 

chat
d-siroma@exp002:~/Desktop/chat/client$ ./a.out exp008 siroma
connected
receive: REQUEST ACCEPTED
receive: USERNAME REGISTERED
daiki >daiki
siroma
siroma >siroma
daiki >Hello
HI!
siroma >HI!
^C
d-siroma@exp002:~/Desktop/chat/client$ 
```

図3 作成したクライアントプログラムとサンプルサーバー間での通信の実行結果

- 図??より正しく通信が行えているとわかる。
- 作成したサーバプログラムとサンプルクライアント間での動作結果

図??に動作結果を示す。

図??の左が作成したサーバープログラム、右がサンプルクライアントプログラムである。サンプルクライアントプログラムのコンソール画面より、他のクライアントが発したメッセージが「ユーザー名 ==> メッセージ」という形で画面に正しく出力されていることがわかる。

また、図??より、ユーザーが離脱した際にそのユーザー名を出力したり、クライアントの離脱や新規接続に関わらずメッセージの送受信が行えていたりすることがわかる。

```

d-siroma@exp001:~/Desktop/chat/server$ gcc -pthread se.c
d-siroma@exp001:~/Desktop/chat/server$ ./a.out
[exp008.exp.ics.es.osaka-u.ac.jp]
user_name: daiki
addr: 4
user_name:
addr: -1
user_name:
addr: -1
user_name:
addr: -1
user_name:
addr: -1
[exp002.exp.ics.es.osaka-u.ac.jp]
user_name: daiki
addr: 4
user_name: siroma
addr: 5
user_name:
addr: -1
user_name:
addr: -1
user_name:
addr: -1
daiki==>Hi
daiki==>I am daiki!
siroma==>hello-
d-siroma@exp008:/home/exp/t-hirai/Public/enshu-c$ ./chatclient exp001 daiki
ICS Exercises C sample program chatclient.c
connected to exp001
join request accepted
user name registered
Hi
I am daiki!
siroma==>hello-
abcde
dddfa
siroma==>what?

d-siroma@exp002:/home/exp/t-hirai/Public/enshu-c$ ./chatclient exp001 siroma
ICS Exercises C sample program chatclient.c
connected to exp001
join request accepted
user name registered
daiki==>Hi
daiki==>I am daiki!
hello-
daiki==>abcde
daiki==>dddfa
what?
d-siroma@exp002:/home/exp/t-hirai/Public/enshu-c$
```

図4 サンプルクライアントと作成したサーバー間での通信の実行結果 1

```

user_name:
addr: -1
user_name:
addr: -1
daiki==>Hi
daiki==>I am daiki!
siroma==>hello-
daiki==>abcde
daiki==>dddfa
siroma==>what?
siroma: exit
[exp008.exp.ics.es.osaka-u.ac.jp]
user_name: daiki
addr: 4
user_name: hana
addr: 6
user_name:
addr: -1
user_name:
addr: -1
user_name:
addr: -1
hana==>haha!!
daiki==>Hi
daiki==>hello
hana==>yes
d-siroma@exp001:~/Desktop/chat/server$ ./a.out
[exp008.exp.ics.es.osaka-u.ac.jp]
user_name: daiki
addr: 4
user_name:
addr: -1
user_name:
addr: -1
user_name:
addr: -1
user_name:
addr: -1
[exp002.exp.ics.es.osaka-u.ac.jp]
user_name: hana
addr: 6
user_name:
addr: -1
user_name:
addr: -1
user_name:
addr: -1
user_name:
addr: -1
[exp008.exp.ics.es.osaka-u.ac.jp]
user_name: siroma
addr: 5
user_name:
addr: -1
user_name:
addr: -1
user_name:
addr: -1
user_name:
addr: -1
d-siroma@exp008:/home/exp/t-hirai/Public/enshu-c$ ./chatclient exp001 daiki
ICS Exercises C sample program chatclient.c
connected to exp001
join request accepted
user name registered
Hi
I am daiki!
siroma==>hello-
abcde
dddfa
siroma==>what?
siroma: connection ended
hana==>haha!!
Hi
hello
hana==>yes

d-siroma@exp002:/home/exp/t-hirai/Public/enshu-c$ ./chatclient exp001 siroma
ICS Exercises C sample program chatclient.c
connected to exp001
join request accepted
user name registered
daiki==>Hi
daiki==>I am daiki!
hello-
daiki==>abcde
daiki==>dddfa
what?
d-siroma@exp002:/home/exp/t-hirai/Public/enshu-c$
```

図5 サンプルクライアントと作成したサーバー間での通信の実行結果 2

- 作成したサーバとクライアント間での動作結果

まず図??に作成したサーバープログラムと作成したクライアントプログラム 5 個を用いた通信の結果を示す。

The screenshot shows five terminal windows on a Linux desktop. The top-left window contains server logs with user names siroma, hana, kai, and ryo, and their corresponding addresses (4, 5, 6, 7). The other four windows show client-side communication between users daiki, kai, siroma, and hana. The clients are sending messages like "aa", "bb", "cc", and "gg" to each other.

```

addr: 4
user_name: siroma
addr: 5
user_name: hana
addr: 6
user_name: kai
addr: 7
user_name:
addr: -1
[exp012.exp.ics.es.osaka-u.ac.jp]
user_name: daiki
addr: 4
user_name: siroma
addr: 5
user_name: hana
addr: 6
user_name: kai
addr: 7
user_name: ryo
addr: 8
daiki==>aa
kai==>bb
kai==>cc
siroma==>gg
daiki==>aa
kai==>bb
kai==>cc
siroma==>gg
daiki==>aa
bb
cc
siroma==>gg
daiki==>aa
kai==>bb
kai==>cc
siroma==>gg

```

図 6 作成したサーバープログラムと作成したクライアントプログラム 5 個を用いた通信の結果

左上がサーバープログラム、右側の上からユーザー名が daiki, siroma, hana, kai, ryo のクライアントプログラムである。この図よりメッセージの送受信が正しく行えていることがわかる。

次に図??にタイムアウト時の結果を示す。サーバープログラムのコンソール画面から daiki, kai, siroma が発言した状態で放置した場合何も発言がなかった hana と ryo が最初に退出し、次に daiki が退出しているとわかる。またあるユーザーが退出した場合、その旨が他のクライアントプログラムのコンソール画面に出力されていることもわかる。

続いて図??に 6 つ目のクライアントプログラムが接続を試みた場合の結果を示す。

図の左下に sin というユーザーが接続を試みているが、リクエストが却下されていることがわかる。

最後に図??にユーザー名が重複している場合の結果を示す。既に daiki, siroma, hana が接続中の状態に新たに daiki というユーザーが接続しようとしているが、ユーザー名の登録リクエストで却下されていることがわかる。またこの図のサーバープログラムのコンソール画面から、使用していない user_name_list は空で、ソケットアドレスは-1 になっていることもわかる。

これらの実行結果から作成したクライアントプログラム、サーバープログラムは要件を満たし正しく機能していることがわかる。

```

Terminal
user_name: hana
addr: 6
user_name: kai
addr: 7
user_name:
addr: -1
[exp012.exp.ics.es.osaka-u.ac.jp]
user_name: daiki
addr: 4
user_name: siroma
addr: 5
user_name: hana
addr: 6
user_name: kai
addr: 7
user_name: ryo
addr: 8
daiki==>aa
kai==>bb
kai==>cc
siroma==>gg
hana: exit
ryo: exit
daiki: exit
[]

Terminal
siroma==>gg
hana==>connection ended
ryo==>connection ended
You don't speak for 30 sec, so you are evicted.
finished
d-siroma@exp009:~/Desktop/chat/client$ []

Terminal
gg
hana==>connection ended
ryo==>connection ended
daiki==>connection ended
[]

Terminal
kat==>cc
siroma==>gg
You don't speak for 30 sec, so you are evicted.
finished
d-siroma@exp004:~/Desktop/chat/client$ []

Terminal
cc
siroma==>gg
hana==>connection ended
ryo==>connection ended
daiki==>connection ended
[]

Terminal
siroma==>gg
hana==>connection ended
You don't speak for 30 sec, so you are evicted.
finished
d-siroma@exp012:~/Desktop/chat/client$ []

```

図7 タイムアウト時の結果

```

Terminal
[exp012.exp.ics.es.osaka-u.ac.jp]
user_name: daiki
addr: 4
user_name: siroma
addr: 5
user_name: hana
addr: 6
user_name: kai
addr: 7
user_name: ryo
addr: 8
daiki==>aaa
siroma==>sss
hana==>ff
user_name: daiki
addr: 4
user_name: siroma
addr: 5
user_name: hana
addr: 6
user_name: kai
addr: 7
user_name: ryo
addr: 8
[]

Terminal
receive: USERNAME REGISTERED
aaa
siroma==>sss
hana==>ff
[]

Terminal
daiki==>aaa
sss
hana==>ff
[]

Terminal
daiki==>aaa
siroma==>sss
ff
[]

Terminal
receive: USERNAME REGISTERED
daiki==>aaa
siroma==>sss
hana==>ff
[]

Terminal
daiki==>aaa
siroma==>sss
hana==>ff
[]

Terminal
d-siroma@exp014:~/Desktop/chat/client$ ls
a.out cl.c
d-siroma@exp014:~/Desktop/chat/client$ ./a.out exp001 sin
connected
receive: REQUEST REJECTED
error: Success
d-siroma@exp014:~/Desktop/chat/client$ []

```

図8 6つ目のクライアントプログラムが接続しようとした結果

The image shows five terminal windows arranged in a grid, illustrating a scenario where multiple users attempt to register with the same username. The terminals are running on a Linux desktop environment.

- Terminal 1 (Leftmost):** Shows two client registrations:

```
user_name:  
addr: -1  
[exp004.exp.ics.es.osaka-u.ac.jp]  
user_name: daiki  
addr: 4  
user_name: siroma  
addr: 5  
user_name: hana  
addr: 6  
user_name:  
addr: -1  
user_name:  
addr: 1  
[exp013.exp.ics.es.osaka-u.ac.jp]  
user_name: daiki  
addr: 4  
user_name: siroma  
addr: 5  
user_name: hana  
addr: 6  
user_name:  
addr: -1  
user_name:  
addr: -1
```
- Terminal 2 (Second from Left):** Shows the first client's registration attempt:

```
connected  
receive: REQUEST ACCEPTED  
receive: USERNAME REGISTERED
```
- Terminal 3 (Second from Right):** Shows the second client's registration attempt:

```
receive: REQUEST ACCEPTED  
receive: USERNAME REGISTERED
```
- Terminal 4 (Third from Right):** Shows the third client's registration attempt:

```
receive: REQUEST ACCEPTED  
receive: USERNAME REGISTERED
```
- Terminal 5 (Rightmost):** Shows the fourth client's attempt to register 'exp001' as 'daiki'. It receives a rejection and then successfully registers as 'siroma'. Finally, it lists the registered users:

```
d-siroma@exp014:~/Desktop/chat/client$ ./a.out exp001 daiki  
connected  
receive: REQUEST ACCEPTED  
receive: USERNAME REJECTED  
error: Success  
d-siroma@exp013:~/Desktop/chat/client$  
daiki==>aaa  
siroma==>sss  
hana==>ff  
^C  
d-siroma@exp012:~/Desktop/chat/client$
```

図9 ユーザーネームが重複している場合

1.3 考察

1.3.1 fork と pthread の違い

当初サーバープログラムを作成し始めた際、指導書をあまり読んでいなかったため fork を用いて、親プロセスで新規クライアントからの接続受付を、子プロセスで各クライアントの処理を行う方針でプログラムを組んでいた。その際に、子プロセスに新規に参加したクライアントや、途中で離脱したクライアントの情報が伝わらなかつたためメッセージの転送がうまくいかないなどの問題が起こった。パイプを用いた通信も行ってみたが記述が多くなりすぎたため断念した。これは fork で分岐される子プロセスは fork 呼び出し時点での情報を参照するからだとわかった。ゆえに fork を用いる場合は、タイムアウトまでの時間を計測するなどのある程度決まった処理、機械的な処理の方が向いており、今回のチャットプログラムではユーザーの参加・離脱が活発で流動性があるためあまり適していないと感じた。一方 pthread は??章でも言及したが、スレッドを生成したタイミングに関わらず、グローバル変数で定義した変数の情報を流動的に参照できるため、まるで複数のプログラムが動いているかようなプログラムを作成できる。

1.3.2 write 関数を連続で呼び出す場合

今回のチャットシステムでは発言がユーザー名 ==> "メッセージ" のように出力される。当初他クライアントへメッセージを転送する際にユーザー名 ==>、メッセージを 3 つに分解し write を 3 回連続して呼び出していたが、そうするとクライアントプログラム側の端末で正しくその内容が表示されなかつた。これは read を再度呼び出すまでにかかる時間が write に対して極めて長いためであるとわかつた。したがって、write を実行する前に、

```
memset(temp, '\0', sizeof(temp));
strcat(temp, user_name_list[client_number]);
strcat(temp, yajirushi);
strcat(temp, rbuf[client_number]);
write(csock[j], temp, sizeof(temp));
```

のように strcat 関数により文字列を結合し、write 関数の呼び出しを 1 回で済むようにしたことで解決した。

2 課題 4-2

2.1 実装した機能

実装した機能は、

- 離脱者を全クライアントの端末に表示させる機能（サーバープログラム）
- 30 秒間発言のないユーザを強制離脱させる機能（クライアントプログラム）

の二つである。

2.2 離脱者を全クライアントの端末に表示させる機能

2.2.1 実装方針とアルゴリズム

クライアントが離脱する場合、そのクライアントを監視するスレッドの read 関数では 0 が返る。その際の処理に該当するコードを以下に示す。

```
char temp[99]; //add
char yajirushi[]=="==>"; //add
char fin_message[]="connection ended\n";
....
```

```

for (int j=1; j<=MAXCLIENTS; j++){
    if(j!=client_number && csock[j]!=-1){
        memset(temp, '\0', sizeof(temp));
        strcat(temp,user_name_list[client_number]);
        strcat(temp,yajirushi);
        strcat(temp,fin_message);
        write(csock[j],temp,sizeof(temp));
    }
}

```

離脱者の情報はメッセージを他のクライアントプログラムに転送する方法と同じである。これによりクライアントが離脱した際に全てのクライアントの端末にその旨を出力できる。

2.2.2 動作確認

図??, 図??を参照。

2.3 30秒間発言のないユーザを強制離脱させる機能

2.3.1 実装方針とアルゴリズム

課題3で実装したタイムアウト機能と基本的には同様の方法を用いた。

該当するコードを以下に示す。

```

void myalarm(int sec) {
    static int pid;
    static int flag = 0;
    if (flag>0){
        kill(pid,SIGTERM);
    }
    flag++;

    if ((pid=fork())== -1) {
        perror("fork failed.");
        exit(1);
    }

    if (pid == 0) { //30秒数える子プロセス
        pid= getpid();
        sleep(sec);
        pid_t p_pid = getppid(); // 親プロセス ID 取得
        kill(p_pid,SIGALRM);
        exit(0);
    }

    signal(SIGCHLD,SIG_IGN);
    return;
}

void timeout()
{
    printf(" You don't speak for 30 sec, so you are evicted.\n");
    finish=1;
}

```

```

...
if(signal(SIGALRM,timeout) == SIG_ERR) {
    perror("signal failed.");
    exit(1);
}
myalarm(TIMEOUT);
if (c4==1){
    do {

        if (finish==1){
            c4=0;
            c5=1;
            break;
        }

        FD_ZERO(&rfds);
        FD_SET(0,&rfds);

...
if(FD_ISSET(0,&rfds)) {
    myalarm(TIMEOUT);
}

```

30秒数える方法は課題3のものと同じであるためここでは説明を割愛する。入力を監視する FD_ISSET が実行される度に myalarm(TIMEOUT) を呼び出すことで、30秒のカウントをリセットしている。逆に他のクライアントのメッセージのやり取りを眺めているだけでは myalarm(TIMEOUT) は呼び出されないため、カウントは続くようになっている。30秒経過すると、関数 timeout により、" You don't speak for 30 sec, so you are evicted.\n"が出力、finish フラグが1になるとより状態 c4 から c5 に遷移し、クライアントプログラムが終了する。

2.3.2 動作確認

図??を参照。

3 発展課題

1章で説明したように、マルチタスクの使用によりサーバープログラムの実装を行なった。

4 感想

自分の直感に従ってプログラムを作成し始めた結果スレッドを用いた開発になり、困難を極めた。しかしその分理解が深まったり、技術力が高またりしたので返って非常に良い経験になった。

付録 A プログラムリスト

A.1 クライアントプログラム

```

1 #include <sys/types.h>
2 #include <sys/socket.h>
3 #include <netinet/in.h>

```

```

4 #include <netdb.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <sys/wait.h>
10 #include <signal.h>
11 #define TIMEOUT 100
12 #define PORT 10000
13 void c6_exception(int);
14 int finish=0;
15
16 void myalarm(int sec) {
17     static int pid;
18     static int flag = 0;
19     if (flag>0){
20         kill(pid,SIGTERM);
21     }
22     flag++;
23
24     if ((pid=fork())== -1) {
25         perror("fork failed.");
26         exit(1);
27     }
28
29     if (pid == 0) { //30 秒数える子プロセス
30         pid= getpid();
31         sleep(sec);
32         pid_t p_pid = getppid(); // 親プロセス ID 取得
33         kill(p_pid,SIGALRM);
34         exit(0);
35     }
36
37     signal(SIGCHLD,SIG_IGN);
38     return;
39 }
40
41 void timeout()
42 {
43     printf(" You don't speak for 30 sec, so you are evicted.\n");
44     finish=1;
45 }
46
47
48 int main(int argc,char **argv)
49 {
50     //s1
51     int sock;
52     int c2=0,c3=0,c4=0,c5=0;
53     int result=0;
54     char buf[1024];
55     struct sockaddr_in host;
56     struct hostent *hp;
57     fd_set rfds;

```

```

58     struct timeval tv;
59
60     if (argc != 3) {
61         fprintf(stderr,"Usage: %s hostname message\n",argv[0]);
62         exit(1); }
63
64     if ( ( sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP) ) < 0) {
65         perror("client: socket");
66         exit(1); }
67
68     bzero(&host,sizeof(host));
69     host.sin_family=AF_INET;
70     host.sin_port=htons(10140);
71
72     if ( ( hp = gethostbyname(argv[1]) ) == NULL ) {
73         fprintf(stderr,"unknown host %s\n",argv[1]);
74         exit(1);
75     }
76
77     bcopy(hp->h_addr,&host.sin_addr,hp->h_length);
78
79     result=connect(sock,(struct sockaddr *)&host,sizeof(host));
80     if ((result)==-1){
81         perror("cannot \n");
82         exit(1);
83     }
84     else{
85         printf("connected\n");
86         c2=1;
87     }
88
89     if(c2==1){
90         //s2
91         memset(buf, '\0', sizeof(buf));
92         if ( ( read(sock,buf,sizeof(buf)) ) < 0) {
93             perror("read");
94         }
95         else {
96             printf("receive: %s\n",buf);
97
98             if( !strncmp( "REQUEST ACCEPTED\n", buf, strlen("REQUEST ACCEPTED\n"))){
99                 c3=1;
100            }
101            // もしメッセージが接続受理 ("REQUEST ACCEPTED\n") ならば状態 c3 へ
102            else c6_exception(sock);
103            //さもなければ（接続拒否または何らかの例外）状態 c6 へ移る.
104        }
105
106
107    }
108
109    //c3
110    if (c3==1){
111        if(write(sock, argv[2], /*sizeof*/strlen(argv[2]))==-1){

```

```

112         perror("writeforsend\n");
113         exit(1);
114     }
115
116     memset(buf, '\0', sizeof(buf));
117
118     if ( ( read(sock,buf,sizeof(buf)) ) < 0) {
119         perror("read");
120     }
121     else {
122         printf("receive: %s\n",buf);
123
124         if( !strncmp( "USERNAME REGISTERED\n", buf, strlen("REQUEST ACCEPTED\n")) ) {
125             c4=1;
126         }
127         // もしメッセージが接続受理 ("USERNAME REGISTERED\n") ならば状態 c4 へ
128         else c6_exception(sock);
129         //さもなければ（接続拒否または何らかの例外）状態 c6 へ移る.
130     }
131 }
132
133
134 //c4
135
136 if(signal(SIGALRM,timeout) == SIG_ERR) {
137     perror("signal failed.");
138     exit(1);
139 }
140
141 myalarm(TIMEOUT);
142 if (c4==1){
143     do {
144
145         if (finish==1){
146             c4=0;
147             c5=1;
148             break;
149         }
150
151         FD_ZERO(&rfds);
152         FD_SET(0,&rfds);
153         FD_SET(sock,&rfds);
154         tv.tv_sec = 1;
155         tv.tv_usec = 0;
156         memset(buf, '\0', sizeof(buf));
157         if(select(sock+1,&rfds,NULL,NULL,&tv)>0 ){
158
159             if(FD_ISSET(sock,&rfds)){
160                 size_t readlen=read(sock,buf,sizeof(buf));
161                 if ( readlen < 0) {
162                     perror("read");
163                     exit(1);
164                 }
165                 else if (readlen == 0) {

```

```

166             c5=1;
167             break;
168         }
169     else {
170         size_t length=strlen(buf);
171         if (length > 0 && buf[length - 1] == '\n') {
172             buf[--length] = '\0';
173         }
174         printf("%s",buf);
175         printf("\n");
176     }
177 }
178
179     if(FD_ISSET(0,&rfds)) {
180         myalarm(TIMEOUT);
181
182         if (fgets(buf, sizeof(buf), stdin) == NULL) {
183             // 標準入力が EOF なら状態 c5 へ
184             c5=1;
185             break;
186         }
187         else{
188             if(write(sock, buf,strlen(buf))==-1){
189                 perror("writeforclient\n");
190                 exit(1);
191             }
192         }
193     }
194 }
195 } while (1);
196 }
197
198 //c5
199 if(c5==1){
200     printf("finished\n");
201     close(sock);
202     exit(0);
203 }
204 printf("exit\n");
205
206 }
207
208
209 void c6_exception(int sock){
210     perror("error");
211     close(sock);
212     exit(1);
213 }
214

```

A.2 サーバープログラム

```
1 #include <sys/types.h>
2 #include <sys/socket.h>
3 #include <netinet/in.h>
4 #include <netdb.h>
5 #include <unistd.h>
6 #include <string.h>
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <sys/wait.h>
10 #include <signal.h>
11 #include <pthread.h>
12 #define MAXCLIENTS 5
13 #define _UNIX03_THREADS
14 int csock[MAXCLIENTS+1]={-1,-1,-1,-1,-1,-1};
15 char rbuf[MAXCLIENTS+1][1024];
16 char user_name_list[MAXCLIENTS+1][1024];
17 int k=0;
18 int connect_check(int);
19 int flag1=0;
20 int flag2=0;
21 int flag3=0;
22 int flag4=0;
23 int flag5=0;
24 pthread_mutex_t mutex;
25
26 void *client_socket(void *arg) {
27     int client_number;
28     char temp[99];
29     char yajirushi[]{"==>"};
30     char fin_message[]{"connection ended\n"};
31     if (flag1!=1) {
32         flag1=1;
33         client_number=1;
34     }
35
36     else if (flag2!=1) {
37         flag2=1;
38         client_number=2;
39     }
40     else if (flag3!=1) {
41         flag3=1;
42         client_number=3;
43     }
44     else if (flag4!=1) {
45         flag4=1;
46         client_number=4;
47     }
48     else {
49         flag5=1;
50         client_number=5;
51     }
```

```

52
53     do {
54         fd_set rfd;
55         struct timeval tv;
56         FD_ZERO(&rfd);
57         FD_SET(0,&rfd);
58         FD_SET(csock[client_number],&rfd);
59         tv.tv_sec = 1;
60         tv.tv_usec = 0;
61         if(select(csock[client_number]+1,&rfd,NULL,NULL,&tv)>0 ){ //s6
62
63             memset(rbuf[client_number], '\0', sizeof(rbuf[client_number]));
64             if(FD_ISSET(csock[client_number],&rfd)) { //s6 , 受信時,
65
66                 size_t readlen=read(csock[client_number],rbuf[client_number],sizeof(rbuf[client_number]));
67                 if ( readlen < 0 ) {
68                     perror("read");
69                     exit(1);
70                 }
71                 else if (readlen == 0) { //ctr+D
72                     //s7
73                     for (int j=1; j<=MAXCLIENTS; j++){
74                         if(j!=client_number && csock[j]!=-1){
75                             memset(temp, '\0', sizeof(temp));
76                             strcat(temp,user_name_list[client_number]);
77                             strcat(temp,yajirushi);
78                             strcat(temp,fin_message);
79                             write(csock[j],temp,sizeof(temp));
80                         }
81                     }
82
83                     printf("%s: exit\n",user_name_list[client_number]);
84                     close(csock[client_number]);
85
86                     memset(user_name_list[client_number], '\0', sizeof(user_name_list[client_number]));
87                     csock[client_number]=-1;
88                     pthread_mutex_lock(&mutex);
89                     k--;
90                     pthread_mutex_unlock(&mutex);
91
92
93                     switch (client_number){
94                         case 1:
95                             flag1=0;
96                             break;
97
98                         case 2:
99                             flag2=0;
100                            break;
101
102                         case 3:
103                             flag3=0;
104                             break;
105

```

```

106         case 4:
107             flag4=0;
108             break;
109
110         case 5:
111             flag5=0;
112             break;
113     }
114 }
115
116 else {
117     write(1,user_name_list[client_number],sizeof(user_name_list[client_number]));
118     write(1,"==>",sizeof("==>"));
119     write(1,rbuf[client_number],sizeof(rbuf[client_number]));
120     //write で他の socket にも出力
121     for (int j=1; j<=MAXCLIENTS; j++){
122         if(j!=client_number && csock[j]!=-1){
123             memset(temp, '\0', sizeof(temp));
124             strcat(temp,user_name_list[client_number]);
125             strcat(temp,yajirushi);
126             strcat(temp,rbuf[client_number]);
127             write(csock[j],temp,sizeof(temp));
128
129         }
130     }
131
132 }
133
134
135 }
136 } while (1);
137 return NULL;
138 }
139
140
141 int main(int argc,char **argv) {
142     int sock;
143     int temp;
144     int reuse;
145     struct sockaddr_in svr;
146     pthread_t handle1;
147     pthread_t handle2;
148     pthread_t handle3;
149     pthread_t handle4;
150     pthread_t handle5;
151     pthread_mutex_init(&mutex, NULL);
152
153     if ((sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))<0) {
154         perror("socket");
155         exit(1);
156     }
157
158     reuse=1;
159     if(setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&reuse,sizeof(reuse))<0) {

```

```

160     perror("setsockopt");
161     exit(1);
162 }
163
164 bzero(&svr,sizeof(svr));
165 svr.sin_family=AF_INET;
166 svr.sin_addr.s_addr=htonl(INADDR_ANY);
167 svr.sin_port=htons(10140);
168
169 if(bind(sock,(struct sockaddr *)&svr,sizeof(svr))<0) {
170     perror("bind");
171     exit(1);
172 }
173
174 if (listen(sock,5)<0) {
175     perror("listen");
176     exit(1);
177 }
178
179 while(1){
180     pthread_mutex_lock(&mutex);
181     if(k>0){
182
183         if (flag1!=1) pthread_create(&handle1, NULL,client_socket, NULL);
184         else if (flag2!=1) pthread_create(&handle2, NULL,client_socket, NULL);
185         else if (flag3!=1) pthread_create(&handle3, NULL,client_socket, NULL);
186         else if (flag4!=1) pthread_create(&handle4, NULL,client_socket, NULL);
187         else if (flag5!=1) pthread_create(&handle5, NULL,client_socket, NULL);
188
189     }
190     pthread_mutex_unlock(&mutex);
191
192     temp=connect_check(sock);
193
194     if(temp!=-1) {
195         pthread_mutex_lock(&mutex);
196         k++;
197         pthread_mutex_unlock(&mutex);
198         for (int i=1;i<MAXCLIENTS+1;i++){
199             if ( csock[i]==-1){
200                 csock[i]=temp;
201                 break;
202             }
203         }
204     }
205     else close(csock[0]);
206     for (int i=1;i<MAXCLIENTS+1;i++){
207         printf("user_name: %s\n",user_name_list[i]);
208         printf("addr: %d\n",csock[i]);
209     }
210 }
211
212 pthread_mutex_destroy(&mutex);
213 }

```

```

214
215 int connect_check(int sock){
216     struct sockaddr_in clt;
217     struct hostent *cp;
218     unsigned int clen;
219     int client_sock=-1;
220     int i,flag=0;
221
222     clen = sizeof(clt);
223     if ( ( csock[0] = accept(sock,(struct sockaddr *)&clt,&clen) ) < 0 ) {
224         perror("accept");
225         exit(2);
226     }
227
228     pthread_mutex_lock(&mutex);
229     if(k<MAXCLIENTS) {
230         write(csock[0], "REQUEST ACCEPTED\n", strlen("REQUEST ACCEPTED\n"));
231
232         cp = gethostbyaddr((char *)&clt.sin_addr,sizeof(struct in_addr),AF_INET);
233         printf("[%s]\n",cp->h_name);
234
235         memset(rbuf[0], '\0', sizeof(rbuf[0]));
236         if ( ( read(csock[0],rbuf[0],sizeof(rbuf[0])) ) < 0 ) {
237             perror("read");
238             exit(1);
239         }
240
241     else{
242         size_t length = strlen(rbuf[0]);
243         if (length > 0 && rbuf[0][length - 1] == '\n') {
244             rbuf[0][-length] = '\0';
245         }
246         for (i=1;i<MAXCLIENTS+1;i++){
247             if(strcmp(rbuf[0],user_name_list[i])==0){
248                 write(csock[0], "USERNAME REJECTED\n", strlen("USERNAME REJECTED\n"));
249                 flag=1;
250                 break;
251             }
252         }
253
254         if (flag!=1){
255             for (i=1;i<MAXCLIENTS+1;i++){
256                 if (user_name_list[i][0]=='\0'){
257                     write(csock[0], "USERNAME REGISTERED\n", strlen("USERNAME REGISTERED\n"));
258                     memset(user_name_list[i], '\0', sizeof(user_name_list[i]));
259                     strcpy(user_name_list[i],rbuf[0]); //register username
260                     client_sock=csock[0]; //register socket
261                     break;
262                 }
263             }
264         }
265     }
266
267 }

```

```
268
269     else{
270         write(csock[0], "REQUEST REJECTED\n", strlen("REQUEST REJECTED\n"));
271     }
272
273     pthread_mutex_unlock(&mutex);
274
275     return client_sock;
276 }
277
```

参考文献

- [1] EYES SOFTWARE, 複数通信先のソケット通信サンプル, <http://www.eyes-software.co.jp/news/archives/7>, 2016.11.03
- [2] Zenn, select 関数を用いた標準入力の監視 [Linux / C 言語], <https://zenn.dev/kedama.nth/articles/13a40615312460>
- [3] ys memos, C++ プロジェクトで pthread 関係のリンクエラーへの対処, <https://ysuzuki19.github.io/post/cpp-cmake-pthread>, 2020/09/21
- [4] 薫の Hack 2022, pthread mutex で排他ロックする方法, https://kaworu.jpn.org/c/pthread_mutex で排他ロックする方法,
- [5] Qiita, [C 言語] pthread/mutex を使った並列処理, https://qiita.com/ryo_manba/items/e48faf2ba84f9e5d31c8, 2021 年 07 月 30 日
- [6] だえうホームページ, [C 言語] 排他制御について解説 [Mutex], https://daeudaeu.com/c_mutex/#Mutex, 2021 年 11 月 23 日