情報科学演習C　課題4レポート

氏名　山久保孝亮
所属　大阪大学基礎工学部情報科学科ソフトウェア科学コース
メールアドレス　u327468b@ecs.osaka-u.ac.jp
学籍番号　09B22084

提出日　2024年8月6日
担当教員　平井健士, 中島悠太

1 課題 4-1

1.1 プログラムの仕様

課題 4-1 で作成したクライアントプログラム chatclient.c の仕様は以下の通りである.

- プログラム実行の書式は"./chatclient [サーバプログラムを実行中のホスト名] [使用したいユーザ名]"である.
- 接続できたかどうか，名前を登録できたかどうかが標準出力に表示される. 接続できた場合はチャット機能が使用でき, 接続できなかった場合はプログラムが終了する.
- チャット機能には以下の 3 つの機能がある.
  1. 標準入力に文字列を書き込んで [Enter] キーを押すと, サーバにその文字列を送信する.
  2. サーバに接続されているほかのホストから送られてきた文字列を受信し, 標準入力に表示させる.
  3. EOF を入力するとサーバとの接続が切れ, プログラムを終了する.

また, サーバプログラム chatserver.c の仕様は以下の通りである.

- プログラム実行の書式は引数はなしである. 即ち"./chatserver"である.
- 一度に接続できるユーザの数は 5 つである.
- 常に新たなユーザの接続を待っており, 新たに接続しようとするユーザが現れた場合には正常に接続できたかどうかと, すでに同じユーザ名が登録されていないかを確認する．どちらも満たしていればその旨の文字列を送信して新たなユーザとして登録し, いずれかを満たさない場合はその旨の文字列を送信して再び待ち状態に入る.
- 登録されたユーザから文字列を受信した場合は，その文字列の先頭にユーザ名を追加して接続中のすべてのクライアントに送信する.
- EOF を受信するとそのユーザの接続を切り, ユーザの情報を削除する.

また，両社のプログラムに共通する仕様は以下のとおりである.

- 使用するポート番号は 10140 である.
- ユーザ名は英数字，ハイフン，アンダースコアのみから成る.
- クライアント側はユーザ名の末尾に改行文字を追加して送信し，サーバ側は受け取った際にその改行文字を取り除いてから保存する.

1.2 クライアントプログラムのアルゴリズム

今回作成したプログラムは指導書の状態に則って作成しており, 以下の図 1 のフローチャートはそれぞれの状態の遷移の様子を表す.

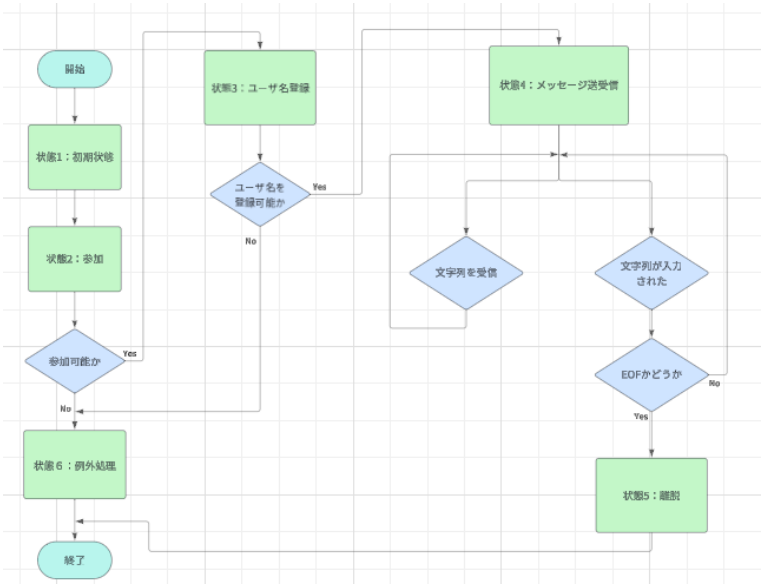

図 1: クライアントプログラムのフローチャート

それぞれの状態は指導書のとおりに実装した．また，図 1 には現れていないが状態 6 の例外処理はプログラム中の操作に問題が発生した際に実行されるので，任意の状態から状態 6 に遷移する．それぞれの状態の概要は以下のとおりである．

状態 1: 初期状態
ソケットを作成し，第一引数のホスト名に接続要求を出す．

状態 2: 参加
サーバから接続できたかどうかの文字列を受け取り，判定する．

状態 3: ユーザ名登録
第二引数のユーザ名を送信し，ユーザ名を登録できたかどうかの文字列を受け取る．そしてそれを判定する．

状態 4: メッセージ送受信
標準入力から文字列が入力されればその文字列をサーバへ送信する．サーバから文字列を受信すればその文字列を標準入力へ出力する．

状態 5: 離脱
標準入力が”EOF”のとき，ソケットを閉じてプログラムを正常終了する．

状態 6: 例外処理
何らかの操作においてエラーが発生した際に，ソケットを閉じてプログラムを異常終了する．

1.3 クライアントプログラムの実装方法

ここではアルゴリズムで記述した各状態とその条件分岐の詳細な実装方法について記述する．以下の表 1 はこのプログラムで使用した変数名とその表す内容である．

変数名	型	表す内容
sock	int	ソケットディスクリプタ
n	int	入出力操作の結果を格納
rbuf	char	送受信する文字列を格納する．要素数は 1024
rfds	fd_set	ファイルディスクリプタのセット
tv	struct timeval	タイムアウト値を設定する構造体
*server	struct hostent	ホスト情報を格納するためのポインタ
svr	struct sockaddr_in	サーバのアドレス情報を格納する構造体
current_time	time_t	現在の時刻を格納
*local_time	struct tm	ローカルタイムを表す構造体へのポインタ
argc	int	引数の個数を格納
*argv[]	char	引数の文字列を格納

表 1: このプログラムで使用した変数

1.3.1 初期状態

状態 1 のプログラムは大きく以下のように処理が分かれる．

1. 書式の確認し, ソケットを作成
2. ホスト名を取得し，接続する

以下でその詳細について記述する．

1. この処理のプログラムは以下のようになる．

```
1 if(argc != 3){
2     fprintf(stderr, "Usage: %s <hostname> <username>\n", argv[0]);
3     exit(1);
4 }
5 if ((sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))<0) {
6     perror("socket");
7     exit(1);
8 }
```

   コマンドライン引数が 3 であるかどうかを確認して書式が正しいかを確認する. ここで argc の値が 3 である理由は”./chatclient”も引数として数えられるためである．そして sock() を使ってソケットを作成する．

2. この処理のプログラムは以下のようになる．

```
1 server = gethostbyname(argv[1]);
2 if (server == NULL) {
3     fprintf(stderr, "ERROR, no such host as %s\n", argv[1]);
4     exit(0);
5 }
6 bzero((char *)&svr, sizeof(svr));
7 svr.sin_family = AF_INET;
8 bcopy((char *)server->h_addr, (char *)&svr.sin_addr.s_addr, server->h_length);
```

```
9  svr.sin_port = htons(10140);
10 if (connect(sock, (struct sockaddr *)&svr, sizeof(svr)) < 0) {状態６の処理
11
12 }
```

コマンドライン引数の１番目の引数をホスト名として変数に格納する．ここでインデックスが１である理由は argv[0] に”./chatclient”が格納されているからである．そしてそのホスト名のサーバがあれば connect() を使って接続をサーバに要請する．

1.3.2 参加

状態２のプログラムは以下のようになる．

```
1  bzero(rbuf, 1024);
2  n = read(sock, rbuf, 1024);
3  if(n <= 0){状態６の処理
4
5  }
6  printf("%s",rbuf);
7  if(strcmp(rbuf,"REQUEST ACCEPTED\n") == 0) {状態３の処理
8
9  }else{状態６の処理
10 }
```

まずサーバから文字列を受信しそれを標準入力に出力する．正常に受信ができればその内容が”REQUEST ACCEPTED \n”であるので，一致しているかどうかを確認し一致していれば状態３へ，一致していなければ状態６へ遷移する．文字列が一致しているかどうかの判定には strcmp() を使用した．

1.3.3 ユーザ名登録

状態３のプログラムは大きく以下のように処理が分かれる．

1. ユーザ名を処理してからサーバへ送信する．
2. サーバからユーザ名の登録に関する文字列を受け取る．

以下でその詳細について記述する．

1. この部分の処理のプログラムは以下のようになる．

```
1  bzero(rbuf,1024);
2  strcpy(rbuf,argv[2]);
3  rbuf[strlen(argv[2])] = '\n';
4  n = write(sock, argv[2], strlen(argv[2]));
5  if(n < 0){状態６の処理
6
7  }
```

まず第二引数のユーザ名をサーバに送信する．引数を送信する際，ユーザ名の終わりを判別するために改行文字を最後に加える．また，rbuf を bzero() を使って初期化してからユーザ名を格納している．これをしていないと，例えば”aiueo”というユーザ名にした場合 rbuf に格納されている文字列が”aiueoST ACCEPED\n”という文字列になってしまうので前の文字列を初期化しなければならない．

2. この部分の処理のプログラムは以下のようになる．

```
bzero(rbuf, 1024);
n = read(sock, rbuf, 1024);
if(n <= 0){状態６の処理

}
printf("%s",rbuf);
```

この処理でも bzero() で rbuf を初期化してから read() を使って文字列を受け取っている．

1.3.4 メッセージ送受信

状態４と待ち状態のプログラムは大きく以下のように処理が分かれる．

1. 待ち状態によるソケットと標準入力の監視
2. 送信処理
3. 受信処理

以下でその詳細について記述する．

1. この処理のプログラムは以下のようになる．

```
while(1){
    FD_ZERO(&rfds);
    FD_SET(0, &rfds);
    FD_SET(sock, &rfds);
    tv.tv_sec = 1;
    tv.tv_usec = 0;
    if (select(sock + 1, &rfds, NULL, NULL, &tv) > 0) {
        if (FD_ISSET(0, &rfds)) {送信

        }
        if (FD_ISSET(sock, &rfds)) {受信

        }
    }
}
```

while 文による無限ループの中で select() を使って標準入力とソケットを監視している．FD_ISET の第一引数が 0 であれば送信，sock であれば受信の処理へ移る．

2. この処理のプログラムは以下のようになる．

```
bzero(rbuf, 1024);
if (fgets(rbuf, 1024, stdin) == NULL) {状態５の処理

}
n = write(sock, rbuf, strlen(rbuf));
if (n < 0) {状態６の処理

}
```

ここでは write() を使ってメッセージの送信の処理を実装している．fgets() によって標準入力から取得された文字列を rbuf へ格納して送信している．このとき，EOF が入力された時は状態５に遷移する．

3. この処理のプログラムは以下のようになる．

```
1 bzero(rbuf, 1024);
2 n = read(sock, rbuf, 1024);
3 if(n <= 0){状態６の処理
4
5 }else{
6     printf("%s",rbuf);
7 }
```

1.3.5 離脱

状態５のプログラムは以下のようになる．

```
1 printf("\nEOF detected\n");
2 close(sock);
3 exit(0);
```

EOF が入力されたことを表示し，ソケットを閉じて exit(0) でプログラムを正常終了する．

1.3.6 例外処理

状態 6 のプログラムは以下のようになる．

```
1 perror("ERROR MESSAGE");
2 close(sock);
3 exit(1);
```

この状態は何らかの処理が正常に動作しなかった際の処理のため，まず perror() を使ってそれぞれの問題に合わせたエラーメッセージを表示する．そしてソケットを閉じて exit(1) でプログラムを異常終了する．

1.4 サーバプログラムのアルゴリズム

今回作成したプログラムは指導書の状態に則って作成しており, 以下の図 2 のフローチャートはそれぞれの状態の遷移の様子を表す.

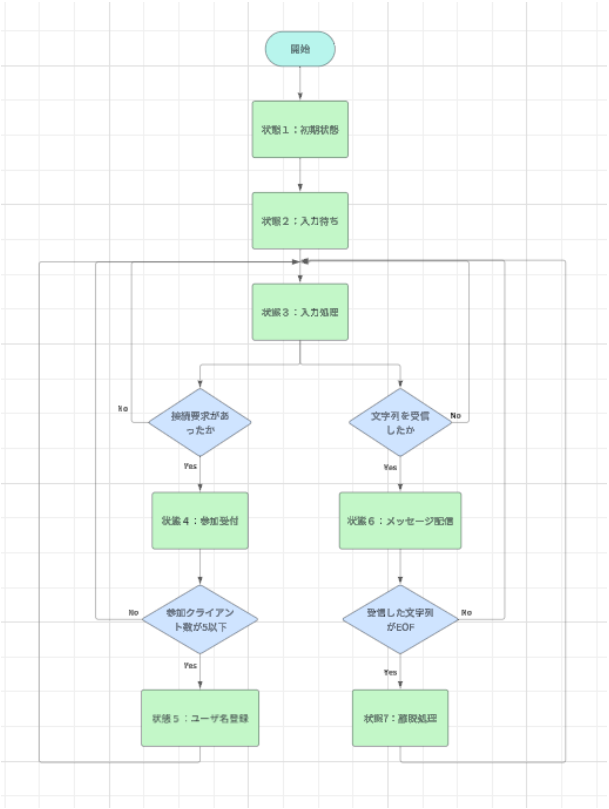

図 2: クライアントプログラムのフローチャート

それぞれの状態は指導書のとおりに実装した．また，図 2 には現れていないが状態 7 の例外処理はプログラム中の操作に問題が発生した際に実行されるので，任意の状態から状態 7 に遷移する．それぞれの状態の概要は以下のとおりである．

状態 1: 初期状態

ソケットを作成し，バインドする．

状態 2: 入力待ち

接続要求とクライアントからの文字列の受信を監視する．

状態 3: 入力処理

接続要求があった場合とクライアントからの文字列の受信があった場合で処理を分岐させる．

状態 4: 参加受付

接続要求を受け付け，待ち受け可能数を超えていなければ登録完了メッセージを送信する．超えている場合は接続拒否のメッセージを送信する．

状態 5: ユーザ名登録

ユーザ名を受信し，現在接続中のクライアントと同じ名前でなければ登録完了メッセージを送信する．同じ名前があれば登録拒否メッセージを送信する．

状態 6: メッセージ配信

受信した文字列の先頭にユーザ名を追加して接続中の全クライアントに対してその文字列を送信する．

状態 7: 離脱処理

何らかの操作においてエラーが発生した際に，ソケットを閉じてプログラムを異常終了する．

1.5 サーバプログラムの実装方法

ここではアルゴリズムで記述した各状態とその条件分岐の詳細な実装方法について記述する．以下の表2はこのプログラムで使用した変数名とその表す内容である．

変数名	型	表す内容
sock	int	ソケットディスクリプタ
n	int	入出力操作の結果を格納
k	int	現在接続されているクライアントの個数を格納
sd	int	クライアントソケットのファイルディスクリプタ
clen	int	クライアントアドレスのアドレス構造体のサイズを格納
max_sd	int	select() で使用するファイルディスクリプタの最大値を格納
new_socket	int	新しいクライアント接続のためのソケットファイルディスクリプタ
new_socket_num	int	新しく接続されたクライアントのソケット番号を格納
username_flag	int	ユーザ名の重複を確認するためのフラグ.
csock	int	接続されたクライアントのファイルディスクリプタを格納する配列
registered_username	char	登録されたユーザ名を格納する配列
svr	struct sockaddr_in	サーバのアドレス情報を格納する構造体
clt	struct sockaddr_in	サーバのアドレス情報を格納する構造体
rfds	fd_set	ファイルディスクリプタのセット
tv	struct timeval	タイムアウト値を設定する構造体
rbuf	char	送受信する文字列を格納する．要素数は 1024
name_message	char	メッセージを構築するためのバッファ
accept_request_message	char	”REQUEST ACCEPTED \n”を格納
reject_request_message	char	”REQUEST REJECTED \n”を格納
accept_username_message	char	”USERNAME ACCEPTED \n”を格納
reject_username_message	char	”USERNAME REJECTED \n”を格納

表 2: このプログラムで使用した変数

1.5.1 初期状態

状態１のプログラムは大きく以下のように処理が分かれる．

1. 配列 csock を全て-1 に初期化
2. 接続のための設定

以下でその詳細について記述する．

1. この処理のプログラムは以下のようになる．

```
1 for (int i = 0; i < MAXCLIENTS; i++) {
2     csock[i] = -1;
3     bzero(registered_username[i],256);
4 }
```

ここでは for 文によりクライアントのファイルディスクリプタを格納するための配列である csock の全要素を-1 に初期化している．これにより，-1 が格納されていればそのインデックスにはクライアントは接続されておらず，-1 以外であればクライアントが接続されていると見分けることができる．

2. この処理のプログラムは以下のようになる．

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

まず socket() でソケットを作成し，bind() で sock に待ち受けするネットワークインターフェイスを割り当てる．そして現在接続されているクライアント数を表す k を 0 に初期化する．

1.5.2 入力待ち

状態２のプログラムは以下のようになる．

```
while(1){
    FD_ZERO(&rfds);
    FD_SET(0, &rfds);
    FD_SET(sock, &rfds);
    tv.tv_sec = 1;
    tv.tv_usec = 0;
    max_sd = sock;
    for(int i=0;i<MAXCLIENTS;i++){
        sd = csock[i];
        if(sd > 0){
            FD_SET(sd, &rfds);
        }
        if (sd > max_sd) {
            max_sd = sd;
        }
    }
    if (select(max_sd + 1, &rfds, NULL, NULL, &tv) > 0) {状態３の処理

    }
}
```

ここでは無限ループの中に select() を使用するための初期化の処理を記述する．また，8 行目からの for 文では csock の内，ソケットディスクリプタが格納されているインデックスを探してそれと現在の max_sd を比較して最大のソケットディスクリプタを格納できるようにしている．そして select() を実行する．

1.5.3 入力処理

状態 3 のプログラムは以下のようになる．

```
1 if (FD_ISSET(sock, &rfds)) {状態４の処理
2
3 }
4 for(int i=0;i<MAXCLIENTS;i++){
5     if ((csock[i] != -1) && (FD_ISSET(csock[i], &rfds))){状態６の処理
6
7     }
8 }
```

1 から 3 行目では接続要求があるかどうかを FD_ISSET() を使って監視している．また，4 から 8 行目では，csock に格納されている登録済みのクライアントからのメッセージの受信を監視している．

1.5.4 参加受付

状態４のプログラムは以下のようになる．

```
1  clen = sizeof(clt);
2  if ((new_socket = accept(sock, (struct sockaddr *)&clt, &clen)) < 0) {
3      perror("accept");
4      exit(2);
5  }
6  if(k<MAXCLIENTS){
7      n = write(new_socket, accept_request_message, strlen(accept_request_message));
8      if(n < 0){
9          perror("ERROR writing");
10         close(new_socket);
11         break;
12     }状態５の処理
13
14 }else{
15     n = write(new_socket, reject_request_message, strlen(reject_request_message));
16     if(n < 0){
17         perror("ERROR writing");
18         close(new_socket);
19         break;
20     }
21     close(new_socket);
22 }
```

1 から 5 行目では accept() を使ってクライアントからの connect() による接続要求を受け入れている．6 から 19 行目の条件分岐では，現在接続中のクライアントの個数 k が 5 より小さいとき，即ちまだクライアントを接続できるときの処理である．このとき，write() で接続が受け入れられた旨の文字列を送信する．12 行目から 18 行目の条件分岐では k が 5 以上，即ちこれ以上クライアントを接続することができないときに

実行される．このとき write() を使って接続が受け入れられなかった旨の文字列を送信し，2 行目で受け入れた new_socket を閉じている．

1.5.5 ユーザ名登録

状態 5 のプログラムの処理は大きく以下のように分けることができる．

1. 登録可能なソケット番号を確認し，登録するユーザ名を取得
2. ユーザ名に関する処理
3. 登録するユーザ名が既に登録されていないかの確認
4. フラグによる分岐とそれぞれの処理

以下でその詳細について記述する．

1. この部分の処理のプログラムは以下のようになる．

```
1  for (int i = 0; i < MAXCLIENTS; i++) {登録可能なソケット番号を確認
2
3      if (csock[i] == -1) {
4          new_socket_num = i;
5          csock[i] = new_socket;
6          break;
7      }
8  }
9  bzero(rbuf, 1024);
10 n = read(new_socket, rbuf, 1024);登録するユーザー名を取得
11
12 if(n <= 0){
13     perror("Connection closed by client.\n");
14     close(new_socket);
15     return 0;
16 }
```

   1 から 8 行目までの for 文では csock のそれぞれの要素が-1 であるか，即ち登録可能な要素が残っているかを調べている．まだ登録可能なものが残っていれば new_socket_num にそのインデックスを格納し，csock のそのインデックス番目の要素にソケットディスクリプタを格納し break する．これにより，一つ登録可能な要素が見つかればこの for 文は終了する．

   9 から 16 行目では rbuf を bzero() で初期化してから read() でクライアントから送信されるユーザ名を取得している．

2. この部分の処理のプログラムは以下のようになる．

```
1  username_flag = 0;
2  for(int i=0;i<1024;i++){
3      if(rbuf[i] == '\n'){
4          rbuf[i] = '\0';
5          break;
6      }
7      if(!isalnum(rbuf[i]) && rbuf[i] != '_' && rbuf[i] != '_'){
8          username_flag = 1;
```

```
9       }
10  }
```

1 行目の username_flag はユーザ名を受け付けるか拒否するかを表すフラグであり，0 は拒否を，1 は受け付けることを表す．2 から 10 行目の for 文では，受信したユーザ名の末尾から改行文字”\n”を取り除き，ユーザ名に英数字とハイフン，アンダーバー以外の文字が含まれていないかを判定している．改行文字の除去は 3 から 6 行目のように文字列の要素を”\n”を比較し一致していればそれをヌル文字に変換することで実現した．指定された文字以外が含まれていないかの判定は 7 から 9 行目のように指定された 3 つの条件全てを満たさないものがあればフラグを 1 にすることで実現した．フラグについての処理は後述する．

また，仕様よりユーザ名は英数字とハイフン，アンダースコアのみから成るのでユーザ名の途中に改行文字が存在し，正確に処理されないということはあり得ない．

3. この部分の処理のプログラムは以下のようになる．

```
1  for(int i=0;i<MAXCLIENTS;i++){
2      if (strcmp(registered_username[i], rbuf) == 0) {
3          username_flag = 1;
4          break;
5      }
6  }
```

for 文を使って登録されたユーザ名を格納している registed_username の全要素と今回新たに受信したユーザ名とを比較し，一致しているものがないかを strcmp() を使って判定する．同じであればフラグを 1 にして break する．

4. この部分の処理のプログラムは以下のようになる．

```
1  if(username_flag == 0){
2      n = write(new_socket, accept_username_message, strlen(accept_username_message))
           ;
3      if(n < 0){
4          perror("ERROR writing");
5          close(new_socket);
6          break;
7      }
8      strcpy(registered_username[new_socket_num], rbuf);
9      printf("%s is registered\n",registered_username[new_socket_num]);
10     csock[new_socket_num] = new_socket;
11     k++;
12 }else{
13     n = write(new_socket, reject_username_message, strlen(reject_username_message))
           ;
14     if(n < 0){
15         perror("ERROR writing");
16         close(new_socket);
17         break;
18     }
19     csock[new_socket_num] = -1;
20     close(new_socket);
21 }
```

1 から 12 行目の条件分岐はフラグが 0，即ちユーザ名が登録可能であるときの処理である．ユーザ名の登録を受け付けた旨の文字列を write() で送信し registered_username の new_socket_num 番目の要素に strcpy() を使ってユーザ名を格納する．また，csock の new_socket_num 番目に新たに登録するソケットディスクリプタ new_socket を格納する．これにより，registered_username と csock のインデックスはそれぞれ対応する．そして最後に接続しているクライアント数を 1 つインクリメントする．

12 行目から 21 行目の条件分岐ではフラグが 1，即ちユーザ名が登録可能ではないときの処理である．ユーザ名の登録を拒否した旨の文字列を write() で送信し，csock の new_socket_num 番目の要素を-1 に，new_socket を閉じる．

1.5.6 メッセージ配信

状態 6 のプログラムは以下のようになる．

```
1  bzero(rbuf, 1024);
2  n = read(csock[i], rbuf, 1024);
3  if(n <= 0){状態 7 の処理
4
5  }else{
6      for(int j=0;j<MAXCLIENTS;j++){
7          if(csock[j] != -1){
8              snprintf(name_message, sizeof(name_message), ">%s %s", registered_username[i
                   ], rbuf);
9              n = write(csock[j], name_message, strlen(name_message));
10             if(n < 0){
11                 perror("ERROR writing");
12                 break;
13             }
14         }
15     }
16 }
```

このプログラムは csock のすべての要素の内のいずれか一つから文字列を受信すると実行される．即ち，このプログラムが処理されるのは csock[i] から文字列を受信したときである．まずクライアント csock[i] から送信された文字列を read() で受け取る．3 から 5 行目の処理はクライアントが EOF を入力したときの処理である．5 から 16 行目は EOF 以外の文字列が入力された時の処理である．ここでは受け取った文字列を接続中のクライアントすべてに送信している．具体的には 6 から 15 行目の for 文の中で以下の処理を実行する．

- csock の値が-1 かどうかを判定して，接続中のインデックスのみに以下の処理を実行する．
- snprintf() を使って先頭に”¿[ユーザ名] [送信する文字列]”を name_message に格納する．ユーザ名は registered_username[i] に格納されている．
- write() を使って name_message を送信する．

1.5.7 離脱処理

状態 7 のプログラムは以下のようになる．

```
close(csock[i]);
printf("%s has left the chat room\n",registered_username[i]);
k--;
csock[i] = -1;
bzero(registered_username[i],256);
break;
```

まず csock[i] を閉じ，i 番目のクライアントが退出した旨のメッセージを標準入力に出力する．次に接続中のクライアント数 k を一つ減らし，csock[i] を-1 に初期化する．また，registerd_username[i] も bzero() で初期化する．以上により，クライアントに関するすべての情報を初期化したので break して再び select() による待ち状態に戻る．

1.6 実行結果

今回の課題 4-1 の実行結果は以下のような場合に分類することができる．

1. 一人だけ適切なユーザ名で接続された時
   あ
2. 二人だけ適切なユーザ名で接続された時
   あ
3. 五人だけ適切なユーザ名で接続された時
4. 3 の状態で新たに適切なユーザ名で接続された時
5.

2 課題 4-2

2.1 プログラムの仕様

2.2 アルゴリズム

2.3 実装方法

2.4 実行結果

3 考察

4 感想

5 謝辞