情報科学演習 C 課題 4 レポート

担当教員：小島 英春
提出者：小山 亮
学籍番号：09B15028
e-mail：u745409b@ecs.osaka-u.ac.jp

提出年月日：平成 29 年 8 月 4 日

1 課題内容

4-1 3 章のプロトコル仕様にしたがって、チャットプログラム（サーバ、クライアント）を作成せよ。チャットでは、3 人以上が参加可能である。また、参加者は途中からでも参加や離脱が可能である。

4-2 課題 [4-1] のプログラムを拡張して、便利と思われる機能を 1 つ以上追加せよ。

2 課題 [4-1]

3 章のプロトコル仕様にしたがって、チャットプログラムを作成した。クライアントプログラムとサーバプログラムに分けて仕様説明する。

2.1 クライアントプログラムの仕様

クライアントプログラムの実行時のコマンド引数は hostname というサーバに username というユーザネームで参加したいとき、以下のように指定する。

% ./chatclient hostname username

コマンドがこの型をみたさない場合は使用法を標準出力に出力し、異常終了する。まず、接続要求を hostname へ送信する。接続要求が受理されれば、 ”REQUEST ACCEPTED\n” が送信されるので、標準出力に ”join request accepted\n” を出力し、次の送信を待つ。接続要求が受理されなければ、 ”REQUEST ACCEPTED\n” ではない文字列が送信されるので、サーバから送信された文字列を標準出力に出力し、プログラムを異常終了する。その後、 username が参加要件を満たしていれば、 ”USERNAME REGISTERED\n” が送信されるので、標準出力に ”username registered\n” を出力し、入力待ち状態に移る。username が参加要件を満たしていなければ、”USERNAME REGISTERED\n” ではない文字列が送信されるので、サーバから送信された文字列を標準出力に出力し、プログラムを異常終了する。入力待ち状態で受け付ける入力は、標準入力からの入力と、サーバから送信される文字列である。受け付けた入力が標準入力である場合、入力された文字列をサーバへ送信する。受け付けた入力がサーバからである場合、入力された文字列を標準出力に出力する。

2.2 サーバプログラムの仕様

サーバプログラムを実行すると、サーバプログラムは入力待ち状態になる。受け付ける入力は、参加していないクライアントプログラムからの接続要求、または、参加しているクライアントプログラムからの文字列である。まず、受け付けた入力が接続要求であった場合、接続要求を受理し、チャットに参加できるかの判断に移る。チャットに参加できるかどうかの判断は、最大参加可能クライアント数をみたすかどうかと、ユーザネームが英数字、ハイフン、アンダースコアのみから成るかと、ユーザネームがすでに使われていないかの 3 点から行う。現在の参加クライアント数が最大参加可能クライアント数未満であれば、接続可能であるとして文字列 ”REQUEST ACCEPTED\n” を送信する。さもなければ、接続不可能であるとして、文字列 ”REQUEST REJECTED\n” を送信する。ユーザネームが英数字、ハイフン、アンダースコアのみから成っており、ユーザネー

ムがまだ使われていない場合は、ユーザネームが登録可能であるとして、文字列 ”USERNAME REGISTERED\n” を送信し、ユーザネームを登録し、参加許可したユーザネームを標準出力に出力して、チャットへの参加が実現される。さもなければ、ユーザネームが登録不可能であるとして、文字列 ”USERNAME REJECTED\n” を送信し、ソケットを閉じ、参加拒否したユーザネームを標準出力に出力する。次に、受け付けた入力が参加しているクライアントからの文字列であった場合、その文字列の内容によって、処理が異なる。入力文字列が EOF であった場合、入力があったソケットを閉じ、対応するユーザネームを標準すつりょくに出力し、クライアントを離脱させる。入力文字列が ”/list\n” であった場合、その入力をしたクライアントに、現在参加しているユーザの名前を送信する。入力文字列が ”/send username message\n” であった場合、username にのみ、message を送信する。入力文字列が ”message\n” であった場合、送信したユーザを含め、参加しているすべてのユーザに message を送信する。

2.3 実行結果

チャットプログラムの機能が実現されているかを確かめるために、さまざまな状況に対して実行結果を観察した。まずは、サーバに対して 3 つのクライアントプログラムで接続し、一人が離脱する状況である。

サーバ

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatserver
accepted : first
accepted : second
accepted : third
disconnected : second
```

クライアント 1

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 first
connected to exp002
join request accepted
user name registered
message
first >message
second >I gotta go to bed
```

クライアント 2

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 second
connected to exp002
join request accepted
user name registered
first >message
I gotta go to bed
second >I gotta go to bed

close
r-koyama@exp002:~/enshuuC/kadai4$
```

クライアント 3

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 third
connected to exp002
join request accepted
user name registered
first >message
second >I gotta go to bed
```

サーバはクライアントの接続要求に対して、受理をして参加したクライアントのユーザネームを標準出力に出力している。クライアントが送信したメッセージは、参加しているすべてのクライアントに送信されている。クライアントが EOF を送信すると、サーバは接続を遮断し、離脱したクライアントのユーザネームを標準出力に出力している。他ののクライアントには離脱したことは通知されない。次に最大参加可能クライアント数である 5 人を超えて接続しようとした場合である。

サーバ

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatserver
accepted : first
accepted : second
accepted : third
accepted : fourth
accepted : fifth
rejected
disconnected : third
accepted : sixth
```

クライアント 6

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 sixth
connected to exp002
join request rejected
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 sixth
connected to exp002
join request accepted
user name registered
```

クライアント 3

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 third
connected to exp002
join request accepted
user name registered

close
r-koyama@exp002:~/enshuuC/kadai4$
```

クライアント first,second,third,fourth,fifth が接続した後に sixthh が接続しようとすると、リクエストは拒否されプログラムは終了する。サーバはユーザネームを受信する前に参加の可否を判断するので、標準出力に参加を拒否したことを出力するが、だれを拒否したかは出力できない。その後クライアント third が離脱すると、クライアント sixth は再び参加可能となっている。

3 課題 [4-2]

課題 [4-1] のプログラムを拡張した。クライアントがサーバへ、”/list\n” という文字列を送ると、参加者のリストが返るという機能と、特定の相手だけにメッセージを送信する機能である。どちらの機能もサーバプログラムの改変のみで実現した。

3.1 ユーザリスト機能

3.1.1 仕様

クライアントがサーバへ文字列を送信した場合は、基本的には、そのまま全ユーザに送信されるが、”/list\n” を送信したときは、送信したクライアントにのみ現在の参加者リストを送信する。

3.1.2 実行結果

サーバ

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatserver
accepted : first
accepted : second
accepted : third
```

クライアント 1

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 first
connected to exp002
join request accepted
user name registered
/list
-----user list-----
first
second
third
--------------------
/listlist
first >/listlist
```

クライアント 2

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 second
connected to exp002
join request accepted
user name registered
first >/listlist
```

クライアント 3

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 third
connected to exp002
join request accepted
user name registered
first >/listlist
```

”/list\n” を送信すると、送信したクライアントにのみ参加者リストが送信されている。 ”/list\n” あとに続けて文字を入力している場合は認識されずただのメッセージとして他のクライアントに送信されている。

3.2 秘密のメッセージ機能

3.2.1 仕様

クライアントがサーバへ文字列を送信した場合は、基本的には、そのまま全ユーザに送信されるが、 ”/send ” で始まる文字列であった場合は、次のスペースまでをユーザネームとし、それ以降の文字列を、指定したユーザネームのクライアントにのみ送信する。

3.2.2 実行結果

サーバ

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatserver
accepted : first
accepted : second
accepted : third
```

クライアント 1

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 first
connected to exp002
join request accepted
user name registered
/send second secret message
/send fourth secret message 2
```

クライアント 2

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 second
connected to exp002
join request accepted
user name registered
first >secret message > second
```

クライアント 3

```
r-koyama@exp002:~/enshuuC/kadai4$ ./chatclient exp002 third
connected to exp002
join request accepted
user name registered
```

”/send ” で始まる文字列を送信すると指定したクライアントにしかメッセージが送信されていない。存在しないクライアントを指定すると、何も起こらない。

4 考察

このプロトコル仕様は言語に依存しない形で書かれているので、Java などで組めば、GUI の実装も比較的簡単に行えるかもしれない。サーバプログラムのプロトコル仕様にあった状態 3 での入

力があったソケット全てに対して処理を終えたら状態 2 へ移るという処理の意味がよくわからなかった。 select に対して、複数の入力が一度になされることがあるということなのか、今回は参加可能クライアント数が 5 という小さい数字であったから、複数の入力が衝突しなかっただけなのか、疑問が残った。

5　感想

はじめ、接続要求を受け付けるソケットと、参加したクライアントのソケットの違いがわからず、悩んだ。また、課題 2 のときに simple-talk-server を複数人と会話ができるように拡張しようとしたが、なかなかうまくいかず、諦めていたプログラムであったので、プロトコル仕様を読みながら、半信半疑でプログラムを組んでいくと、進まないこともあったが、意外に簡単に組めたので、感動した。