情報科学演習 C レポート 4

藤田 勇樹
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
学籍番号: 09B16068
メールアドレス: u461566g@ecs.osaka-u.ac.jp
担当教員
小島 英春 助教授
内山 彰 助教授

提出日: 2018 年 8 月 3 日

1　チャットクライアント

クライアントの実装について述べる．基本的にプロトコルの状態遷移に沿った説明を行うが，実際のプログラムでは逐次処理やループで実装できたため，状態変数を用意して分岐するといった厳密な状態遷移は行っていない．サーバの実装も同様.

1.1　状態 c1 —初期状態—

ここでは，サーバへの接続を行う．`socket()` でソケットを生成し，接続先情報を設定し，`connect()` で接続を行う．

1.2　状態 c2 —参加—

`read()` でサーバからソケットを通じて 17 バイトだけメッセージを受信する[*1]．この内容が `"REQUEST ACCEPTED\n"`であれば，接続が受理されたことになる．それ以外であった場合，接続が受理されていないためエラー処理である c6 へ移行する．

1.3　状態 c3 —ユーザ名登録—

引数で指定されたユーザ名に改行を付与し，ソケットに `write()` する．その後，`read()` で 20 バイト受信し，内容が`"USERNAME REGISTERED\n"`であれば，ユーザ名が受理されたことになる．そうでない場合，c6 へ移行する．

1.4　状態 c4 —メッセージ送受信—

システムコール `select()` を用いて，標準入力とソケットからの入力を同時に監視し，標準入力からの入力はソケットに `write()` してサーバに送信，ソケットからの入力は標準出力に出力する．なお，標準入力からの入力が `EOF`(Ctrl-D) であった場合は代わりに状態 c5 に移行する．それ以外の場合は c4 を繰り返す．

1.5　状態 c5 —離脱—

`close()` でソケットを閉じてサーバとの通信を終了し，プログラムを終了する．

1.6　状態 c6 —例外処理—

`close()` でソケットを閉じてサーバとの通信を終了し，プログラムを異常終了する．

[*1] 資料のプロトコルでは文字列の終端の NULL 文字を必須としていないため，NULL 文字は付与されていないことを前提にしている．代わりに，今回作成したプログラムでは基本的に改行文字を文字列の終端に付与するようにした．`<string.h>`のライブラリ関数を使用する場合は，一部を除いて，受信したメッセージの最後のバイトを NULL 文字に置き換えて C 文字列としている．

1.7 拡張機能

1.7.1 プロンプト表示

状態 c4 でキーボードからの入力を受け付ける際，プロンプトとして “(ユーザ名) >” を表示するようにした．これにより，自分の入力が分かりやすくなる．ただし，これだけだと自分の送ったメッセージがサーバから返される際に同じ表示が 2 つ続いてしまうので，プロンプトと入力した内容は送信時に端末上の表示を消去するようにした．ただし，**??**節で述べる`/list` や`/send` から始まるメッセージはそのままサーバから返ってくることはないので，その場合に限り端末上の表示は消去しない．

2 チャットサーバ

2.1 ソケットとユーザ名の管理

ソケットとユーザ名の管理にはリスト構造を用いた．構造体 socklist を以下で定義する．

```
struct socklist {
    int csock;
    char username[101];
    struct socklist* next;
}
```

csock にはクライアントと通信するソケットのファイルディスクリプタを，username にはユーザ名を，next には次の socklist へのポインタを格納する．

また，これとは別にグローバル変数 sockhead と csocknum を定義し，それぞれリストの先頭とクライアント数 k を表す変数とした．

2.2 状態 s1 —初期状態—

`socket()` でソケットを生成し，接続先情報を設定し，`bind()` と `listen()` で接続要求を待機するようにする．

2.3 状態 s2 —入力待ち—

接続要求を待機するソケットとクライアントと通信するソケットすべてからの入力を `select()` で同時に監視する．監視するファイルディスクリプタ集合 `rfds` はクライアントが参加/離脱する度に更新する必要があることに注意する．

2.4 状態 s3 —入力処理—

`select()` で監視した結果，接続要求を待機するソケットからの入力であれば状態 s4 へ，クライアントからの入力であれば状態 s6 へ移行する．

その後，状態 s2 へ移行する．

2.5 状態 s4 —参加受付—

accept() でソケットからの接続要求を受理する．csocknum が MAXCLIENTS 以下なら接続を受理したことを表す"REQUEST ACCEPTED\n"を受け付けたクライアントに送信し，クライアントリストに追加 (newsock とする)，csocknum を 1 増やし，状態 s5 へ移行する．csocknum が MAXCLIENTS を超えていた場合は，接続を拒否したことを表す"REQUEST REJECTED\n"を送信し，受け付けたソケットを閉じ, 状態 s3 へ移行する．

2.6 状態 s5 —ユーザ名登録—

次にユーザ名 + 改行の形でユーザ名が送信されるため，read() で受信し，クライアントリストに同じユーザ名のものがあれば登録拒否とし，"USERNAME REJECTED\n"を送信してソケットを閉じ，newsock をクライアントリストから削除し，csocknum を 1 減らして状態 s3 に移行する．同じユーザ名がなければ，newsock の username を受信したユーザ名にし，"USERNAME REGISTERED\n"を送信する．その後状態 s3 に移行する．

2.7 状態 s6 —メッセージ配信—

select() が反応したクライアントソケットに対し，"read()"でメッセージを受信する．内容が EOF(read() の返り値が 0) なら状態 s7 に移行する．それ以外の場合，受信したメッセージの先頭にユーザ名を追加し"username >message"の形にして，クライアントリストの全てのソケットに"write()"する．その後状態 s3 に移行する．

2.8 状態 s7 —離脱処理—

EOF を送信したクライアントのソケットを"close()"し．クライアントリストから該当する要素を削除，csocknum を 1 減らし，状態 s3 に移行する．

2.9 拡張機能

2.9.1 ユーザ名リスト

状態 s6 において，受信したメッセージが"/list"で始まっていれば，そのメッセージを送信したクライアントにその時点で接続しているユーザの一覧を送信する．

2.9.2 秘密のメッセージ

状態 s6 において，受信したメッセージが"/send username message"の形であれば，ユーザ名が username のクライアントにのみ，message の内容を送信する．このとき，送信するメッセージは，"(送信元のユーザ名)*>message"とし，通常のメッセージと区別する．ユーザ名が username であるクライアントが接続していなかった場合や，"/send"から始まっているが形式が不適切な場合，送信元に使用方法を送信する．

2.9.3 参加，離脱の通知

クライアントが参加/離脱した時, 全クライアントに`"Server >(user) joined/exited:(users) online"`を送信する．(user) には参加/離脱したクライアントのユーザ名が，(users) には参加中のクライアント数が入る．

2.9.4 サーバ終了の予告

サーバに Ctrl-C を入力しプログラムを終了しようとすると，その時点で参加しているクライアントにあと 10 秒で終了する旨を送信し，その 10 秒後にプログラムを終了する．この間に再び Ctrl-C が入力された場合，この処理を最初からやり直す (最後に Ctrl-C を入力してから 10 秒で終了する).

3 実行結果

以下，exp046 でサーバを，exp051 と exp064 でクライアントを実行する．

クライアント数が MAXCLIENTS を超える時の動作を確認する．chatserver の MAXCLIENTS を 1 に設定した．

4 考察

5 感想