情報科学演習 C 課題 4

文責： 内山 彰 *

2018 年 7 月 2 日 (月)

1 目的

複数クライアントから成るサーバ・クライアント型システムの作成法を学ぶ。

2 複数ユーザ間の会話サービスを提供するシステム — チャットシステム

チャットシステムとは、複数人がリアルタイムで 1 行程度のメッセージをやり取りするシステムのことである。チャットシステムは図 1 のように構成される。まずあらかじめどこかのホストでチャットサーバを動かしておく。チャットの参加者はそれぞれチャットクライアントを用いて、サーバが動いているホストの指定されたポートに接続する。接続すると、ユーザ名などの登録処理の後、同じサーバに接続している他の参加者との会話 (チャット) が楽しめる。

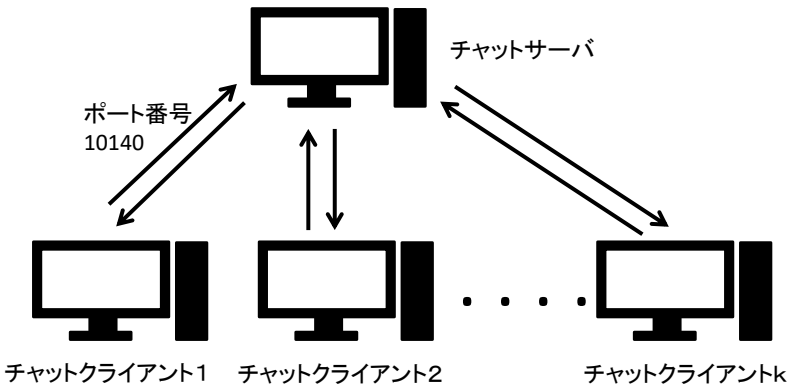

図 1: チャットシステムの構成

各参加者の発言は、チャットサーバを経由してすべての参加者の端末に発言者の名前つきでコピーされる (図 2 参照)。課題 2 で作成した 1 対 1 通信を実現する簡易 talk と異なり、チャットでは一般に 3 人以上の参加が可能である。さらに、参加者は途中からでも自由にチャットに参加あるいは離脱が可能である。

*本稿は、小島先生の書かれた 2017 年度演習 C 資料課題 4 を基にして作成した。

```
...
foo> こんばんわ。
bar> いらっしゃい。＞foo さん
hoge> ようこそ。＞foo さん
...
bar> そろそろ寝るので落ちます。＞all
hoge> ばいばい ＞bar さん
foo> おやすみなさい ＞bar さん
...
```

図 2: チャットクライアントの出力例

3 チャットプロトコルの仕様

チャットシステムを実現する通信手順には様々な物が考えられる。しかし、各人が勝手に決めた手順に従ってプログラムを作成したのでは、他の人が作ったクライアントやサーバと通信できない。そうならないためには、チャットを実現する通信の規約 (プロトコル) をきちんと決めておく必要がある。通信規約を記述した文書を**プロトコル仕様**という。一般にプロトコルを実装するプログラム言語は C 言語であるとは限らないため[1]、仕様には純粋に通信の手順に関係する事項のみを記述すべきである。ゆえに、プロトコル仕様はより抽象的な状態機械モデル (あるいはそれに類するモデル) で記述されることが多い。そこで、本課題でも状態機械モデルで記述したプロトコル仕様を示す。

- 同時接続可能なクライアント数を `MAXCLIENTS` とする。本課題では `MAXCLIENTS`= 5 とする[2]。
- 使用するポート番号は **10140** とする。
- サーバのホスト名および参加するときのユーザ名はクライアントのコマンド引数で以下のように指定するものとする。

  ```
  % chatclient (ホスト名) (ユーザ名)
  ```

- ユーザ名は英数字、ハイフン (“-”) およびアンダースコア (“_”) のみから成る文字列とする。

以下にクライアント側およびサーバ側のプロトコル仕様をそれぞれ示す。() 内にヒントとして使用するシステムコール名を示す。

3.1 クライアント側のプロトコル仕様

状態 c1 (初期状態)

[1] C++,Java,perl,ruby などの言語でもソケットを用いた通信プログラムを作成できる。
[2] 定員オーバーによる接続拒否の処理をテストするため、`MAXCLIENTS` の値は少なめに設定している。

ソケットを生成し (socket)、引数で指定されたホストへ接続要求を出す (connect)。

状態 c2 へ移る。

状態 c2 (参加)

サーバから 17 文字のメッセージを受信する (read) 。

もしメッセージが接続受理 (`"REQUEST ACCEPTED\n"`) ならば状態 c3 へ、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。

状態 c3 (ユーザ名登録)

引数で指定されたユーザ名 (最後に改行) を送信する。(write)

続いて 20 文字の文字列を受信し (read)、

受信文字列が登録完了メッセージ (`"USERNAME REGISTERED\n"`) であれば、状態 c4 へ、さもなければ、エラーメッセージを表示し、状態 c6 へ移る。

状態 c4 (メッセージ送受信)

標準入力とサーバ両方の入力を待ち (select)、

標準入力からの文字列はサーバへ送信し (write)、

サーバからの受信文字列は標準入力へ出力する (read)。

標準入力が EOF なら状態 c5 へ、さもなければ状態 c4 へ移る。

状態 c5 (離脱)

ソケットを閉じて (close)、

クライアントプログラムを正常終了する (exit(0))。

状態 c6 (例外処理)

ソケットを閉じて (close)、

クライアントプログラムを異常終了する (exit(1))。

3.2 サーバ側のプロトコル仕様

状態 s1 (初期状態)

ソケットを生成 (socket) し、接続要求を待つように設定する (bind,listen)。参加クライアント数 k を 0 に初期化する。

状態 s2 へ移る。

状態 s2 (入力待ち)

接続要求を待つソケットからの入力、および、参加しているすべてのクライアントのソケットからの入力[3]を一定時間監視する (select)。

[3]最初は参加クライアント数 $k = 0$ なので、接続要求を待つソケットだけを監視することになる。本プロトコルでは途中からの参加も可能にするため、一般に既存クライアントの入力と新しいクライアントの接続要求を同時に監視する。

状態 s3 へ移る。

状態 s3 (入力処理)

入力があったすべてのソケットに対して入力処理を行う。

接続要求あり: 状態 s4 へ移る。

クライアント $i(i = 1, \ldots, k)$ から入力あり: 状態 s6 へ移る。

入力があったソケットすべてに対して処理を終えたら状態 s2 へ移る。

状態 s4 (参加受け付け)

接続要求を受理する (accept)。

次に、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") を返し (write)、

クライアントソケット[4]を $k+1$ 番目のクライアントとして登録し、状態 s5 へ移る。

さもなければ接続拒否のメッセージ (文字列"REQUEST REJECTED\n") を返し (write)、

クライアントソケットを閉じて (close)、

状態 s3 へ移る。

状態 s5 (ユーザ名登録)

“ユーザ名＋改行” を受信し (read)、1〜k 番目のユーザのいずれかと同じ名前ならば、登録拒否のメッセージ (文字列"USERNAME REJECTED\n") を返し (write)、登録拒否したユーザ名を分かりやすく標準出力に出力し、クライアントソケットを閉じて (close)、状態 s3 へ移る。

未登録のユーザならば、

$k+1$ 番目のクライアントのユーザ名として登録する。登録完了メッセージ (文字列"USERNAME REGISTERED\n") を送信し (write)、登録したユーザ名を分かりやすく標準出力に出力し、

k を 1 増やす。状態 s3 へ移る。

状態 s6 (メッセージ配信)

クライアント i から文字列を入力する (read)。

もし EOF ならば状態 s7 へ移る。

さもなければその入力文字列の先頭にユーザ名を追加して、全クライアントに対して送信し (write)、

状態 s3 へ移る。

[4] accept() で返されるソケットのことである。

状態 s7 (離脱処理)

クライアント i のソケットを閉じる (close)。

離脱したユーザ名を分かりやすく標準出力に出力し、クライアント数 k を 1 減らすなどの離脱処理を行い、状態 s3 へ移る。

4 課題 4-1

3 章に示したプロトコル仕様にしたがってチャットプログラム (クライアントおよびサーバプログラム) を作成せよ。作成したチャットプログラムは、サンプルプログラムとの通信が正しくできなければならない。

サンプルプログラム (サーバおよびクライアント) のコンパイル済みの実行ファイル (バイナリ) はディレクトリ`/home/exp/staff/utiyama/public/enshu-c_sample` に`chatserver`, `chatclient` の名前で置いている。`chatclient` を引数なしで実行すれば、コマンドの使い方が表示される。

仕様を守ったプログラムを作成しないと、他人が作成したプログラムと通信できないことになるので、仕様には厳格に従うこと。例えばメッセージの文字列`"REQUEST ACCEPTED\n"`や、送受信の順序などは必ず仕様通りにしなければならない。サンプルサーバと作成したクライアント、作成したサーバとサンプルクライアント、作成したサーバとクライアント間で通信が正しくできることを必ず確認すること。

逆にいえば、それらの通信に関する手順が仕様に従っているならば、例えば繰り返し命令として for 文を使おうが while 文を使おうが各人の自由である。また、プロトコルに関する部分以外 (例えば、各クライアントのソケットおよびユーザ名を管理するデータ構造など) は仕様に書いていないので、各人が創意工夫すること。

なお、課題 3 で演習した fork を使ったマルチプロセスを用いる実装は大変難しいため、ここでは課題 2 で作成した select を用いた全二重通信プログラムを拡張する形での実装を推奨する。課題 3 で学んだマルチタスク方式で実装する場合は、下記発展課題の 3 つめを参照のこと。

5 課題 4-2

chatserver,chatclient を拡張し、以下に示すような便利と思われる機能を追加せよ。他の機能拡張を考えても良い。前回の課題までは機能拡張は発展課題であったが、今回は必修なので、最低 1 つは機能拡張を行うこと。やった数が多いほど、また、難しいものをやるほど加点が高い。

- 各発言の先頭にユーザ名以外に、発言時刻やクライアントの IP アドレスなどの情報が表示される機能
- 同一のユーザ名のユーザが既に接続している場合にはユーザ名を変更してからの再接続を求める機能

- サーバへ"/list\n"という文字列を送ると現時点の全参加者のユーザ名リストが返されるという機能
- サーバへ"/send hoge message\n"という文字列を送ると、ユーザ名 hoge の端末にだけ message が表示されるというように、特定の相手だけに秘密のメッセージを送信する機能
- 参加や離脱が生じたら、誰が参加したか、誰が離脱したか、現在の参加者総数は何人であるなどといった情報が全参加者の端末に表示される機能
- クライアント起動時にサーバのアドレスを入力せずとも、サーバを自動的に探し出して接続する機能[5]
- サーバの終了をユーザに予告する機能。サーバ側で [CTRL]+c が入力された場合に、終了する旨をユーザにメッセージで通知してから終了する機能[6]。即座に終了するのではなく、一定時間、ユーザに残り時間を通知しつつ待機してから終了すれば、なおユーザに優しいサーバとなる。
- [難] 一定時間発言のないユーザを強制離脱させる機能
- [難] プログラムのユーザインターフェースを考慮し、使い易いものにしてみよ。例えば X ウィンドウシステムを用いて GUI を用意したり、といったことである。

6 発展課題

余力のある者は以下の課題に挑戦してみよ。

- chatserver,chatclient を UDP を用いるように改造せよ。UDP 版チャットシステムの実行結果を示し、チャットシステムは UDP を用いるのにふさわしいアプリケーションといえるか考察せよ。
- クライアント側に [CTRL]+c で利用可能な特殊機能を実装せよ。シグナルハンドラを使って SIGINT を処理することで、[CTRL]+c を押してから何らかの操作を受け付けるようにし、課題 4-2 に挙げたような"/list\n"機能や"/send hoge message\n"機能が実行できるように chatclient を拡張する。[CTRL]+c を押してからの操作方法は色々な方法が考えられる。例えば、[CTRL]+c 後にコマンドを入力する、[CTRL]+c でメニューが表示されるのでそれを選ぶ、などである。
- マルチタスクによる実装を行ってみよ。ただし、演習 3 で用いた fork によるマルチプロセスで実装することは難しい。この例の場合は、pthread_create により起動でき、よりシステムに対する負荷の小さいスレッドを用いることが簡単である。main 関数内では、新たなユーザからの接続を accept し、プロトコルに従ったメッセージ

[5] いくつかの実装方法が考えられる。UDP を併用し、以前の拡張課題にあったブロードキャストを用いるのも手段の 1 つである

[6] 課題 3 で用いた signal システムコールで SIGINT をトラップすると、[CTRL]+c が押された際にプログラムを終了する代わりに指定された関数が実行されるようになる

のやりとりでユーザ名の登録に成功した場合には、リストに新ユーザの情報 (名前、socket) を追加し、各ユーザ用のスレッドを起動する処理のみを繰り返せばよい。各ユーザ用のスレッドは、ユーザの発言を他のユーザへ転送する処理のみをループで繰り返し、ユーザが接続を切った場合には、リストから情報を削除してスレッドを終了する[7]。

マルチスレッドを用いる場合は、グローバル変数などの内容変更が他のスレッドからも同様に参照できるため、スレッド間の情報のやりとりに pipe を使った通信は必要ではない。例えば、上記の設計の場合、現在接続中のユーザと対応する socket のリストをグローバル変数とした配列に保持しておく方針が考えられる。各ユーザ用のスレッドでメッセージを受け取った場合は、そのリストに記載された socket へ順にメッセージを転送すればよい。しかし、同時に複数のスレッドから同一のグローバル変数を書き換えた場合には正しく動作しない可能性があるため、クリティカルセクションとして保護する必要がある。この例では、リストの内容が書き換えられている最中に、別のスレッドからリストを参照・変更しようとすると不具合が起こる。そのため、ユーザの追加・削除・メッセージの転送、はクリティカルセクションとする必要がある。

マルチタスクにより実装した場合には、ソースコードの全体構造は遙かに単純にすることが出来る反面、クリティカルセクションの制御などを適切に行う必要が出てくる。クリティカルセクションは課題 3 で用いたセマフォでももちろん実装可能だが、スレッドの場合は pthread_mutex という同様の仕組みも利用可能である。

pthread_create やその他必要となるシステムコールに関して調べ、実装を行ってみよ。

- ソケット通信機能が利用できる他の言語 (Java,perl など) で chatclient(もし可能なら chatserver も) を作成し、C 言語で作成したものと混在させて通信させてみよ。課題 4-2 の機能拡張を盛り込んでも良い。

- 今までの課題で作成したアプリケーション以外の、クライアント・サーバ型のアプリケーション (データベース、ネットワーク対戦ゲームなど) を自由に設計し、好きなプログラム言語で実装せよ。設計したプロトコルの仕様もきちんと記述すること[8]。

7 Tips

7.1 文字列比較について

課題 4 では受信文字列と指定された文字列を比較する処理がよく現れるが、受信文字列には文字列最後の'\0'(ヌル文字) は含まれないことに注意せよ。read() で受信したデータに対して単純に strcmp() で比較しても一般に正しい結果は得られない。

[7] exit システムコールで終了するプロセスとは違い、スレッドは関数から抜けることで終了できる

[8] レポートの採点では、作成したプログラムの優劣よりも、プロトコルや実装に関する説明が正しく行えているかどうかを重視する。

7.2 EOF の入力と判定

ターミナルの行頭で [CTRL]+d を押すと EOF になる。ターミナルに 1 文字以上を入力している時に [CTRL]+d を押しても、EOF を入力したことにはならないので注意せよ。また、stdio.h で定義されている EOF マクロは整数値であり char 型ではないので、EOF と char 型を比較しないよう注意すること。EOF の判定方法に関するミニレクチャ資料も参考にすること。

7.3 EOF の送信

これまでの課題で既に使っているように、ソケットを close すると通信相手に EOF が送られる[9]。EOF そのものは文字ではないため、文字列と同様の方法では送信できないことに注意せよ。

8 レポート提出について

- 課題 4 レポート
  サーバ、クライアントプログラムのソースコード及び **PDF 形式のレポート**を CLE を介して提出してもらう。ソースコードは，レポートとは別に提出すること。レポートの表紙は 1 ページを使って、
  - 授業名
  - 課題名
  - 学籍番号
  - 名前

  を書くこと。レポートの内容に関しては、少なくとも、
  - 実行結果（サンプルサーバと作成したクライアント、作成したサーバとサンプルクライアント、作成したサーバと作成したクライアント、それぞれの実行結果を示すこと）
  - 考察、工夫した点、強調したい点
  - 感想、意見、疑問等 (演習 C 全体の感想も歓迎します)

  等を盛り込むこと。もちろんその他にもレポートに書くべきことはあるが、それについては、演習の説明会の時に配布した資料等を参考にして作成すること。

  最終レポート提出の締め切りは、
  - 8/3(金)23:59

  とする。

[9] 他にも方法はあるので調べてみると良い。