

# 情報科学演習C 課題2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6月3日  
担当教員 平井健士, 中島悠太

## 1 課題 2-1

### 1.1 アルゴリズム

今回私が作成した echoclient プログラムは以下の図 1 ようなフローチャートとなる.

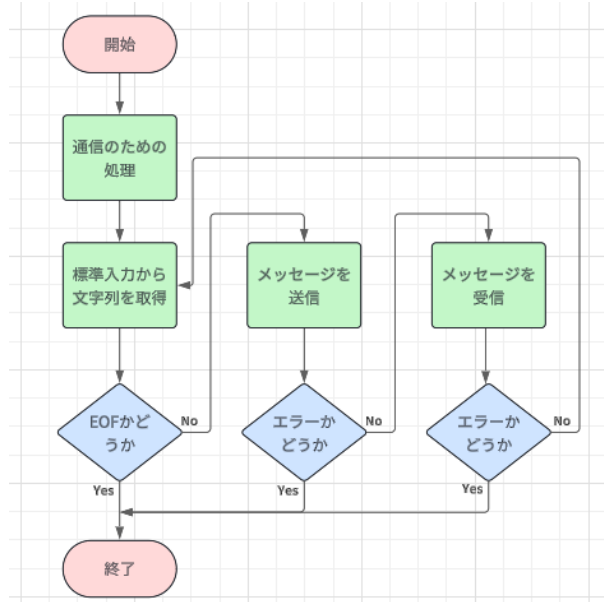


図 1: echoclient プログラムのフローチャート

また, 通信のための処理の内容は以下の通りとなる

1. 入力形式の確認
2. ソケットの生成
3. ホストが存在するかどうかを確認
4. ソケットアドレス再利用の指定
5. ソケット接続先の情報設定
6. ソケットをサーバーに接続

以下でその詳細について述べる.

#### 1.1.1 入力形式の確認

今回の echoclient プログラムは第一引数としてホスト名を指定するという使用法を想定している. そのため, それ以外の使用を制限するために main 関数のコマンドライン引数である argc の値を確認することで実現した. argc は指定した引数の個数+1 の値を表すのでこれが 2 出ない場合は正しい書式でないとしてエラーメッセージを表示し, プログラムを終了する.

### 1.1.2 ソケットの生成

ソケットを生成するには,socket システムコールを使用する. 今回は UDP を使用するので第一引数には AF\_INET, 第二引数には SOCK\_DGRAM, 第三引数には 0 を指定した. 第三引数に 0 を指定するとドメインの種類とソケットの型によってデフォルトのプロトコルが選ばれる.[2] socket システムコールは正常に実行された場合は負でないソケット記述子を返し, 以上があれば-1 を返す. そのため, 正常に終了したかどうかを確かめるために返り値を格納する int 型変数 sock の値が 0 より小さいかどうかを条件分岐で判定する. 正常に実行されていないければエラーメッセージを出力しプログラムを終了する.

### 1.1.3 ホストが存在するかどうかを確認

ホストが存在するかどうかを確認するために gethostbyname システムコールを使用した. 引数は一つでホスト名を指す文字列で, 正常終了すると hostent 構造体へのポインタを返し, エラーが発生した場合は NULL を返す.[1] したがって, hostent 構造体の構造体変数のポインタとして server を定義し, これに gethostbyname システムコールの返り値を格納した. また, 引数にはコマンドライン引数の argv[1] を使用した. その後 gethostbyname システムコールが正常に終了したかどうかを判定するために server に NULL ポインタが返されていないかを条件分岐によって確認する. もし返されていればエラーメッセージを出力しプログラムを終了する.

### 1.1.4 ソケットアドレス再利用の指定

setsockopt 関数を使ってソケット関連のオプションを設定する. 引数は 5 つあり, いかにもその詳細を記述する.[3][4]

- 第一引数: オプションの適用先であるソケット
- 第二引数: オプションが設定しているレベル. 様々なレベルがあるが, SOL\_SOCKET は第一引数で指定したソケットのオプションのレベルが得られる.
- 第三引数: 指定するソケットオプションの名前. 様々なオプションがあるが, 今回使用する SO\_REUSEADDR について記述する. これは通常一定時間ほかのソケットがそのポートを使えなくなってしまうことを防ぐことができるようになる.[5]
- 第四引数: オプションデータへのポインタ. SO\_REUSEADDR を有効にするには整数型で 1 の値が渡される.
- 第五引数: オプションデータの長さ.

また, 返り値は正常に実行された場合は 0 を返し, されなかった場合は-1 を返す. したがって, 今回作成したプログラムでは int 型変数 reuse を 1 に設定し, setsockopt 関数の引数に使用して正常に実行されたかを確認するために返り値が 0 未満かどうかで条件分岐をさせた. 正しく実行されていない場合はエラーメッセージを出力しプログラムを終了した.

### 1.1.5 ソケット接続先の情報設定

ここでは, hostnet 構造体と sockaddr\_in 構造体のメンバに情報を格納している. sockaddr\_in 構造体のメンバは以下のようにになっている.[6][7]

- sin\_len: sin\_addr の変数のサイズ

- `sin_family`：アドレスファミリを指定
- `sin_port`：ポート番号を指定
- `in_addr sin_addr`：IP アドレスを `in_addr` 構造体で指定
- `sin_zero[8]`：OS の内部仕様専用

`sockaddr_in` 構造体の構造体変数名は `svr` とした。以下のプログラムのように情報を設定した。1 行目では `svr` のすべてのメンバを `bzero` 関数を使ってゼロに初期化している。2 行目は `svr` のメンバ `sin_family` に `AF_INET` を設定している。3 行目ではサーバーの IP アドレスを 1.1.5 で取得し、`server` のメンバに格納されているサーバーの IP アドレスを `svr` のメンバに `bcopy` 関数を使ってコピーする。4 行目では `svr` のメンバにポート番号を指定している。

### 1.1.6 ソケットをサーバーに接続

ここでは `connect` システムコールを使ってソケットをサーバに接続している。`connect` システムコールの引数は以下になる。[8]

- `socket`：ソケット記述子
- `address`：接続試行先の `sockaddr` 構造体へのポインター
- `address_len`：`address` パラメータがさす `sockaddr` 構造体のサイズ

また、返り値は正常に終了した場合は 0 を、異常が発生すると -1 を返す。今回のプログラムでは第一引数に `sock`、第二引数に `svr` へのポインタ、第三引数に `svr` 構造体の長さを格納している。そして `connect` システムコールの返り値が 0 より小さければ異常が発生したとしてソケットを閉じてプログラムを終了する。

### 1.1.7 繰り返し処理

今回のプログラムの仕様では標準入力から読み込んだ文字列をサーバプログラムへ送信し受信した文字列を EOF を受け取るまで繰り返すというものであった。したがって、以下の 1.1.8 と 11.9 で記述するメッセージの送信と受信は EOF が入力されるまで繰り返し実行される必要があるから、`while` 文を使ってそれらの処理を無限ループにした。つまり、各ループごとにユーザに `fgets` 関数を使って標準入力から文字列を配列 `rbuf` に格納し、それをサーバに送信、受信するというアルゴリズムを採用した。

### 1.1.8 メッセージをサーバーに送信

ここでは `write` システムコールを使ってソケットにデータを送信している。`write` システムコールの引数は以下になる。[9]

- `fs`：ソケット記述子
- `buf`：書き込まれるデータを保留するバッファを指すポインタ
- `n`：`buf` パラメータが指すバッファの長さ

返り値は、正常に実行されたときは 0 以上の値を返し正常に実行されなかった場合は -1 を返す。`fs` には `sock`, `buf` には `rbuf`, `n` には `strlen` 関数を使って取得した `buf` の長さを設定した。そして変数 `n` に `write` システムコールの返り値を格納する。`n` が 0 より小さければ異常に終了したと判定しソケットを閉じてプログラムを終了する。

### 1.1.9 メッセージをサーバーから受信

ここでは read システムコールを使ってソケットからデータを読み取っている.read システムコールの引数は以下になる.[10]

- fs : ファイルまたはソケットの記述子
- buf : データを受け取るバッファへのポインタ
- n : buf パラメータがさすバッファの長さ

返り値は, 正常に実行されたときは読み込んだバイト数を, 異常終了した場合は-1 を返す.[11] 今回のプログラムでは fs に sock,buf に rbuf,n には strlen 関数を使って取得した buf の長さを設定した. そして変数 n に read システムコールの返り値を格納する. n が 0 より小さければ異常に終了したと判定しソケットを閉じてプログラムを終了する.

## 1.2 実行結果

以下の図 2 と図 3 は echoserver と echoclient を実行し, 文字列”aiueo”を入力したときの動作結果を表している.

```
k-yamakb@exp029:~/enshuC$ ./echoserver
[exp011.exp.lcs.es.osaka-u.ac.jp]
closed
```

図 2: echoserver の実行結果

```
k-yamakb@exp011:~/enshuC$ ./echoclient exp029
enter message:aiueo
aiueo
enter message:
EOF detected
```

図 3: echoclient の実行結果

echoserver は exp029 というホストで,echoclient は exp011 というホストでプログラムを実行した. 仕様のとおり,client 側から文字列を入力すると改行されて正しく同じ文字列が表示されていることがわかる. そして最後に EOF を入力するとサーバとの接続が終了している.

## 2 課題 2-2

### 2.1 simple-talk-server のアルゴリズム

今回私が作成した simple-talk-server プログラムのフローチャートは以下の図 4 のようになる.

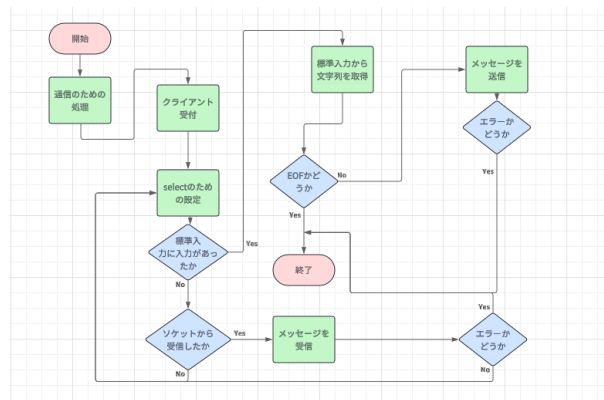


図 4: simple-talk-server プログラムのフローチャート

simple-talk-server の通信のための処理は以下のとおりである。

1. ソケットの生成 (1.1.2 と同様)
2. ソケットアドレスの再利用の指定 (1.1.4 と同様)
3. クライアント受け付け用ソケットの情報設定
4. ソケットアドレスの割り当て
5. 待ち受けクライアント数の設定

以下でその詳細について述べる。

### 2.1.1 クライアント受付用ソケットの情報設定

基本的には 1.1.5 で述べた内容と同じであるが、このプログラムの 3 行目ではサーバ側の IP アドレスを指定している。htonl 関数は引数がホストバイトオーダーで表現された 32 ビット数で、返り値は TCP/IP ネットワークバイト順で値を返す。[12]INADDR\_ANY とするとサーバは全ての利用可能なインターフェースからの接続を受け付けるということを意味する。[13]

### 2.1.2 ソケットアドレスの割り当て

ここでは bind システムコールを使用してソケットをサーバのアドレスにバインドしている。bind システムコールの引数は以下になる。[14]

- sockfd：ファイルディスクリプタ
- addr：sockaddr 構造体のポインタで関連付けるアドレス情報を保持
- addrlen：addr のアドレス構造体のサイズを指定

返り値は正常に実行されると 0 が返され、エラーが発生した場合は -1 が返される。今回のプログラムでも sockaddr\_in 構造体の構造体変数名を svr としているため、第一引数に 2.1.2 で取得した sock、第二引数に svr のアドレス、第三引数に svr のアドレスのサイズを指定している。エラーを検出するために bind 関数の返り値が 0 より小さいときにはプログラムを終了する。

### 2.1.3 待ち受けクライアント数の設定

ここでは listen システムコールを実行してソケットを接続待ちソケットとする。[15] 引数は以下の 2 つである。

- sockfd：ファイルディスクリプタ
- backlog：sockfd についての保留中の接続のキューの最大長

返り値は成功した場合は 0 が返され、エラー時には -1 が返される。今回のプログラムでは第一引数に 1.1.2 で取得した sock、第二引数には 5 を指定する。エラーを検出するために listen 関数の返り値が 0 より小さいときにはプログラムを終了する。

#### 2.1.4 クライアント受付

1 行目では `svr` とは別に定義された `sockaddr_in` 構造体の構造体変数である `clt` を使用して `clt` のサイズを `clen` に格納する。これは、`clt` が接続してきたクライアントのアドレス情報を格納するためのものであるからである。2 行目では `accept` システムコールの戻り値を `csock` に格納する。1.1.2 で取得した `sock` ではなく `csock` を使う理由は、`accept` は待ち受けている接続要求の一つを受け入れ、新しいファイルディスクリプタを返すためである。`accept` システムコールの引数は以下のとおりである。

- `s`: `listen` システムコールによって待ち受け状態となっているソケットを識別するファイルディスクリプタ
- `sockaddr *addr`: 通信相によって認識されている接続中エントリのアドレスウを受け取るバッファへのポインタ
- `*addrlen`: `addr` パラメータがさす構造体の長さを指す変数へのポインタ

戻り値は成功したときは新しいソケットのファイルディスクリプタを、失敗すると `-1` を返す。したがって、失敗したかどうかを判定するために `accept` 関数の戻り値が格納されている `csock` が `0` より小さい時にはプログラムを終了する。

#### 2.1.5 クライアントのホストの情報取得

ここでは `gethostbyaddr` 関数を使って IP アドレスからホスト情報を逆引きしている。これにより、2.1.6 で受け付けたクライアントの情報を取得する。また、ここまで処理が進めばクライアント側と接続したことになるので”Connected with (クライアント側のホスト名)”を出力する。

#### 2.1.6 select のための設定

今回のプログラムの仕様では、サーバ側もクライアント側も標準入力に入力があった時とメッセージを相手側から受信したときとで処理する内容が違っていた。この機能を実現するために `select` システムコールを使用した。これを使うことによりファイルディスクリプタの状態の変化を監視することができる。

ここでは `select` システムコールを使うための設定を行っている。1 行目の `FD_ZERO` は `fd_set` 型の `rdfs` を空集合にするマクロで、2,3 行目の `FD_SET` はファイルディスクリプタである `0` と `csock` を `rdfs` に追加している。ファイルディスクリプタを `0` にすると、標準入力であることを表す。<sup>[16]</sup> これにより、標準入力空の入力があるかどうかとクライアントソケットからのデータ受信を監視できるようになる。4 から 5 行目では監視する待ち時間を `timeval` 構造体のメンバの値に設定している。`timeval` 構造体のメンバは以下のとおりである。

- `tv_sec`: 指定する時間の 1 秒以上の部分
- `tv_usec`: 指定する時間の 1 秒未満の部分

即ち、今回のプログラムでは構造体変数名を `tv` として待ち時間を 1 秒に指定している。

#### 2.1.7 繰り返し処理

`select` システムコールの引数は以下のとおりである。<sup>[1]</sup>

- 監視するファイルディスクリプタの最大値に 1 を加えた整数。`NULL` ポインタを入れると監視しない
- 読み込みを監視するファイルディスクリプタの集合。`NULL` ポインタを入れると監視しない

- 書き込みを視するファイルディスクリプタの集合.NULL ポインタを入れると監視しない
- 例外発生視するファイルディスクリプタの集合.NULL ポインタを入れると監視しない
- select を監視する待ち構造体 timeval へのポインタ.NULL を指定すると監視するファイルディスクリプタのいずれかの状態が変化するまで無期限にブロックする。

監視しない項目は NULL ポインタを入れる。返り値は状態が変化したファイルディスクリプタの総数または-1 を返す。今回のプログラムでは引数を csock+1,rfds のアドレス,NULL,NULL,tv のアドレスを指定している.select が 0 より大きい値を返す時には標準入力または読み込み可能なファイルディスクリプタが存在し,0 以下であればタイムアウトが発生したことを表し,これを無限ループの中に入れることによって常に監視し続けることができる。

### 2.1.8 メッセージをサーバーに送信

FD\_ISSET を使って 0 が rfds に入っているかどうかを判定する。入っていない場合は 0 を返し,入っている場合はそれ以外の値を返す.[17] 基本的な処理内容は 1.1.8 と同じであるが,エラーが発生した場合はソケットを close してプログラムを終了するようにする。

### 2.1.9 メッセージをサーバーから受信

FD\_ISSET を使って csock が rfds に入っているかどうかを判定する。基本的な処理内容は 1.1.9 と同じであるが,エラーが発生した場合はソケットを close してプログラムを終了するようにする。

## 2.2 simple-talk-client のアルゴリズム

今回私が作成した simple-talk-server プログラムのフローチャートは以下の図5のようになる。また,simple-

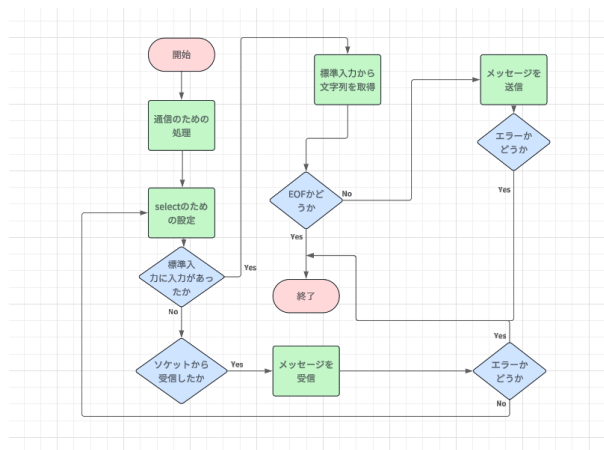


図 5: simple-talk-client プログラムのフローチャート

talk-client の通信のための処理は以下のとおりである。

1. 書式的確認
2. ソケットの生成 (1.1.2 と同様)
3. ホストが存在するかどうかを確認



4. ソケットアドレス再利用の指定 (1.1.4 と同様)
5. サーバ受付用アドレスの設定 (1.1.5 と同様) ただし, ポート番号は 10130 とした.
6. ソケットをサーバに接続 (1.1.6 と同様)

また,select のための処理以降は simple-talk-server と同じ処理である. 以下では,simple-talk-server とは異なっている 1 と 3 内容の詳細を記述する.

### 2.2.1 書式の確認

このプログラムは引数にホストを持つので, コマンドライン引数の長さである argc を使って引数の個数による条件分岐を作成した. 正しい書式である,argc が 2 の時以外はプログラムを終了するものとした.

### 2.2.2 ホストが存在するか確認

ここでは, 第一引数で指定したホストが存在するかを gethostbyname システムコールを使って確認している.1.1.3 で述べたように, 正常終了すると返回值として hostent 構造体へのポインタを返し, エラーが発生したときは NULL を返すので返回值を格納している変数 server が NULL かどうかの条件分岐を作成し, エラーの場合即ち引数で指定したホストが見つからない場合はプログラムを終了する.

## 2.3 実行結果

以下の図 6 と図 7 は simple-talk-server と simple-talk-client を実行し, サーバから文字列”aiueo”を, クライアントから文字列”kakikukeko”を入力したときの動作結果を表している.

```
k-yamakb@exp029:~/enshuC$ ./simple-talk-server
connected with [exp011.exp.ics.es.osaka-u.ac.jp]
aiueo
kakikukeko
EOF detected
```

図 6: simple-talk-server の実行結果

```
k-yamakb@exp011:~/enshuC$ ./simple-talk-client exp029
connected
aiueo
kakikukeko
Connection closed by client.
: Success
```

図 7: simple-talk-client の実行結果

echoserver は exp029 というホストで,echoclient は exp011 というホストでプログラムを実行した. 仕様のとおり, お互いの文字列を正しく表示していることがわかる. そして最後に EOF を入力するとサーバとの接続が終了している

## 3 発展課題

### 3.1 lowerechoserver の作成

echoserver のコードの一部を書き換えて実装した. 受信した文字列中のすべての大文字を小文字に変換するために ctype.h というインクルードファイルをインクルードし,write システムコールの直前で受信した文字列が格納されている rbuf の各要素に対して tolower 関数を適用しその返回值を変換した後の文字列を格納する配列 lbuf に格納する. これは while 分の繰り返しにより行われ, 繰り返しの条件は rbuf の要素がヌル文字であるかどうかである. ヌル文字であれば繰り返しを終了し,lbuf の次の要素の内容をヌル文字にし,write システムコールによって lbuf を出力する. 以下の図 8 と図 9 は lowerechoserver と echoclient を実行し, クライアントから文字列”aiueo”と”Aiueo”を入力したときの動作結果を表している.

```
k-yamakb@exp011:~/enshuC$ ./lowerechoserver
[exp029.exp.ics.es.osaka-u.ac.jp]
closed
```

図 8: lowerechoserver の実行結果

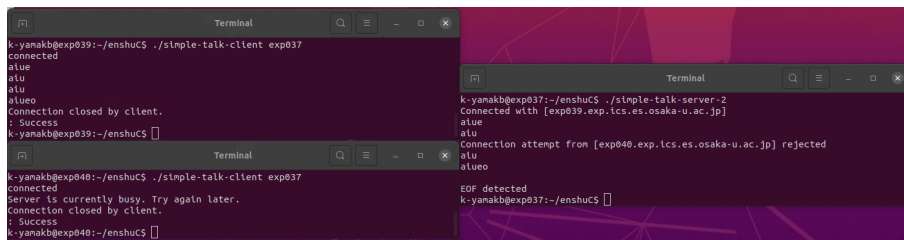
```
k-yamakb@exp029:~/enshuC$ ./echoclient exp011
enter message:aiueo
aiueo
enter message:Aiueo
aiueo
enter message:
EOF detected
```

図 9: echoclient の実行結果

echoserver は exp011 というホストで,echoclient は exp029 というホストでプログラムを実行した。仕様のとおり,client 側から文字列を入力すると大文字を小文字に変換して正しく同じ文字列が表示されていることがわかる。そして最後に EOF を入力するとサーバとの接続が終了している。

### 3.2 起こりうる例外的な動作

このプログラムは simple-talk-server-2 という名前で作成した。simple-talk-server において、一つのサーバに複数のクライアントが接続しようとしたときには、最初に接続されたクライアントとの接続は継続し、それ以外のクライアントからの受付は拒否するという仕様に変更することによってこの例外的な動作へのエラー処理を実現した。具体的な実装方法としては,select システムコールに新しい接続要求があったかどうかを判定する条件分岐を追加して検出し、現在混んでいるという文字列を新たに接続しようとしてきたクライアントに対して送信する。以下の図 10 は exp039 がサーバである exp037 と接続した後さらに exp040 がサーバに接続しようとした様子を示している。



```
k-yamakb@exp039:~/enshuC$ ./simple-talk-server-2
connected
aiueo
aiueo
Connection closed by client.
: Success
k-yamakb@exp039:~/enshuC$

k-yamakb@exp037:~/enshuC$ ./simple-talk-client exp037
connected
aiueo
aiueo
Connection attempt from [exp040.exp.ics.es.osaka-u.ac.jp] rejected
aiueo
aiueo
EOF detected
k-yamakb@exp037:~/enshuC$

k-yamakb@exp040:~/enshuC$ ./simple-talk-client exp037
connected
Server is currently busy. Try again later.
Connection closed by client.
: Success
k-yamakb@exp040:~/enshuC$
```

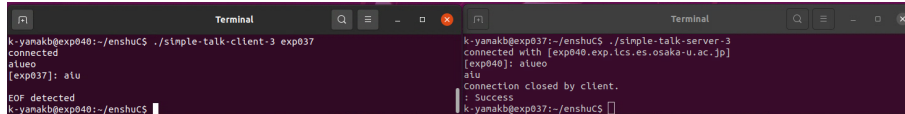
図 10: 実行結果

上図から分かるように、接続要求を出したクライアントには現在混みあっているという文字列が表示され、サーバにはどのクライアントから接続要求があったのかが表示される。また、この接続の拒否が起こった後でも課題 2 のようにメッセージの送受信は正常に実行されている b。

### 3.3 課題 2-2 のプログラムの改良

このプログラムは simple-talk-server-3, simple-talk-client-3 という名前で作成した。メッセージに誰の発言かを表示させるために送信するメッセージの内容に送り主のホスト名を追加するという方針で実装した。具体的には,hostname という配列を用意し,gethostname 関数を使って hostname に自分のホスト名を格納する。そして送信する際の処理のところで,fgets 関数によってメッセージが格納された rbuf と hostname を snprintf 関数を使って”[ホスト名]:メッセージの内容”の形で結合した文字列を配列 sbuf に格納する。この sbuf を write システムコールで送信することにより使用を実現した。これはクライアント側もホスト側も同じように実装した。

図 11 から分かるようにホスト exp037 がサーバプログラムを、ホスト exp040 がクライアントプログラムを実行しており、双方向からのメッセージは受信したときにはお互いのホスト名を表示していることがわかる。



```
k-yanakb@exp040:~/enshuC$ ./simple-talk-client-3 exp037
connected
aluo
[exp037]: alu
EOF detected
k-yanakb@exp040:~/enshuC$

k-yanakb@exp037:~/enshuC$ ./simple-talk-server-3
connected with [exp040.exp.lcs.es.osaka-u.ac.jp]
[exp040]: aluo
alu
Connection closed by client.
: Success
k-yanakb@exp037:~/enshuC$
```

図 11: 実行結果

## 4 感想

今回の課題を通して UDP 及び TCP への理解が深まり, またプログラムを使ってどのようにサーバーとクライアントが接続されるのかを理解することができた. 個人的にネットワークには興味があるので, 今後の課題も興味を持ちながら取り組んでいきたいと思う.

## 5 考察

今回の課題を通して私が考察したことは複数のクライアントの受付を可能にするということである. 今回課題として実装したのは 1 つのホストと 1 つのクライアントのメッセージ交換であるが, 1 つのホストと 2 つ以上の複数のクライアントを接続するためには csock などの接続したソケットを配列とし, 全ての配列の要素に格納されているファイルディスクリプタに対して select を用いて監視することによって機能を実現するのではないかと考えた. また二つ以上のソケットを接続するために while 分による繰り返しを課題 2-2 のプログラムの繰り返しのさらに外側に配置する必要があると考えた.

## 6 謝辞

今回の課題を通して質問対応, レポート採点等をしてくださった教授, TA の皆様方ありがとうございました. 今後の課題もよろしくお願いいたします.

## 参考文献

- [1] 情報科学演習 C 指導書
- [2] <https://www.ibm.com/docs/ja/zos/2.5.0?topic=functions-socket-create-socket> 5/21 アクセス
- [3] <https://www.ibm.com/docs/ja/zos/2.4.0?topic=functions-setsockopt-set-options-associated-socket> 5/21 アクセス
- [4] <https://docs.oracle.com/cd/E19455-01/806-2730/sockets-49/index.html> 5/21 アクセス
- [5] <https://qiita.com/bamchoh/items/1dd44ba1fbef43b5284b> 5/21 アクセス
- [6] <http://www.fos.kuis.kyoto-u.ac.jp/le2soft/siryo-html/node16.html> 5/28 アクセス
- [7] <http://cms.phys.s.u-tokyo.ac.jp/~naoki/CIPINTRO/NETWORK/struct.html> 5/28 アクセス
- [8] <https://www.ibm.com/docs/ja/zos/2.2.0?topic=functions-connect-connect-socket> 5/27 アクセス

- [9] <https://www.ibm.com/docs/ja/zos/2.5.0?topic=functions-write-write-data-file-socket> 5/27 アクセス
- [10] <https://www.ibm.com/docs/ja/zos/2.5.0?topic=functions-read-read-from-file-socket> 5/27 アクセス
- [11] <https://cgengo.sakura.ne.jp/read.html> 5/27 アクセス
- [12] <https://chokuto.ifdef.jp/advanced/function/htonl.html> 5/28 アクセス
- [13] [https://qiita.com/tajima\\_taso/items/2f0606db7764580cf295](https://qiita.com/tajima_taso/items/2f0606db7764580cf295) 5/28 アクセス
- [14] <https://www.man7.org/linux/man-pages/man2/bind.2.html> 5/28 アクセス
- [15] <https://manpages.ubuntu.com/manpages/focal/ja/man2/listen.2.html> 5/28 アクセス
- [16] <https://qiita.com/toshihiroock/items/78286fccf07dbe6df38f> 5/29 アクセス
- [17] <https://docs.oracle.com/cd/E19253-01/819-0392/sockets-20/index.html> 5/29 アクセス