

ネットワーク実験レポート課題

学正番号：09B21601 氏名：西澤陽

提出日 2024/1/22

1 プログラムの処理方針と作成方針

本実験で作成プログラムはサーバー側とクライアント側でデータの通信を行うプログラムである。

1.1 作成方針

本プログラムではサーバー側とクライアント側を分けて別々に作成する。初めにクライアント側について作成する。目標は Web サーバーに対して、HTTP を用いて接続し、Web コンテンツを取得することである。

クライアント側が完成したら、サーバー側を作成する。目標は既に作成したクライアントと1度に一つの通信可能なサーバーを作成することである。

サーバーとクライアントが完成したら、名簿管理プログラムをサーバー、クライアント通信へ拡張する。通信の概要はサーバー側で名簿プログラムを動かし、クライアントから送られてくるコマンドをサーバー側で実行し、結果をクライアントに返すシステムである。

1.2 処理方針

本プログラムの処理方針について述べる。初めにクライアントで通信相手の情報を取得し、情報からソケットを作成する。その間にサーバー側では待ち受けソケットを作成し、ソケットを待ち受け設定にする。サーバー側で待ち受けを開始する。その後、クライアントが TCP コネクション確立を試みる。それに対して、サーバーはコネクションを受け入れ、クライアントとサーバー同士でデータの送受信を行う。最後に共にデータ通信ソケットを終了させ、通信が終了する。

2 プログラム処理の説明

本章ではプログラム処理をソースコードと併せて解説する。クライアントとサーバーの順番で解説する。

2.1 クライアント

始めに通信相手について調べる処理を行う。このとき用いるのが `getaddrinfo` 関数である。初めに最低限の設定をした構造体 `addrinfo hints` を作成する。構造体のメンバを本実験では以下のように設定した。

ソースコード 1: `addrinfo hints` の設定

```
1  memset(&hints,0,sizeof(hints));
2  hints.ai_socktype = SOCK_STREAM;
3  hints.ai_family = AF_INET;
```

```

4  if(getaddrinfo(argv[1],argv[3],&hints,&res) !=0){
5      printf("error:getaddrinfoargv[0]= %s\n",argv[1]);
6  }

```

hints を 0 埋めし、2 行目でソケットタイプを TCP に設定する。3 行目で受け入れるプロトコルファミリーを IPv4 に設定している。以上の設定と、通信相手の IP アドレスもしくはドメイン名、接続ポート番号を決定した getaddrinfo 関数で通信相手の情報を取得する。取得した情報は res に格納され、以降はこれを使う。

次は通信を確立する工程について説明する。socket 関数に res に格納された通信の前提となる情報を与え、ソケットを作成する。通信を確立するために connect 関数にソケット、通信相手の IP アドレス等を与える。通信を確立しようとした時、サーバー側で待ち受け状態であると接続要求が受け入れられ通信が始まる。

通信が始まると、クライアントでは標準入力で名簿管理プログラムを動かすためのコマンドを受け付ける。コマンドを 0 埋めしたバッファ send_mes に格納し、サーバー側に送信する。標準入力から送信までのプログラムを以下に示す。

ソースコード 2: 標準入力とその送信

```

1  memset(send_mes,0,sizeof(send_mes));
2  n = read(0,send_mes,256);
3  if(n== -1 || n==0){
4      printf("ファイルの読み込みエラー\nat:inToOut");
5      exit(0);
6  }
7  ~省略~
8  if(send(s,send_mes,BUF_SIZE,0)==-1)
9  {
10     fprintf(stderr,"sendERROR\n");
11     return 0;
12 }

```

memset 関数で send_mes を 0 埋めしている。標準入力を read 関数で行い send_mes に結果を格納している。read 関数のエラー処理は 3 から 6 行目で行っている。7 行目で send 関数にソケット、バッファとそのサイズを渡しサーバー側へバッファ内容を送信している。send 関数にてエラーが起きると、戻り値が -1 となるため以降の行でエラー処理をしている。送信が完了するとサーバー側から応答があり、recv 関数で受信し、myprint で表示する。ソケットの作成から、受信内容を表示までの一連の処理をクライアントが終了するまで繰り返す。省略箇所は %Q が入力されたときの記述である。

なお %Q が入力された場合のみクライアントを終了させる仕様があるため、別にプログラム記述した。

ソースコード 3: %Q が入力された時の処理

```

1  int next_procec(char *input){
2      //printf("%d:next_procec\n",strlen(input));

```

```

3     if (strcmp(input, "%Q") == 0) {
4         return 0;
5     }else if(input[0]=='%'){
6         return 1;
7     } else {
8         printf("一致する文字列がありません\n");
9         return -1;
10    }
11 }
12 // %Q の処理
13 if(next_procec(send_mes)==0)
14 {
15     printf("クライアント終了\n");
16     if(send(s,err_mes,BUF_SIZE,0)==-1)
17     {
18         fprintf(stderr,"sendERROR\n");
19         return 0;
20     }
21     recv(s,buf,BUF_SIZE,0);
22     close(s);
23     break;
24 }

```

next_procec 関数は send_mes の中身が %Q か確認する関数である。%Q なら 0 を返し、13 から 24 行目の処理を行う。ここでは send 関数で err_mes を送信する。err_mes には”NON”という文字列が格納されており、特に文字列に意味はない。クライアントの送信に対する応答を受け取ったのち 22 行目の close 関数でソケットを削除している。

2.2 サーバー

次にサーバーでの処理の説明をする。初めに socket 関数でソケットを作成する。IP プロトコルは IPv4、トランスポートプロトコルは TCP のため引数は以下のように設定した。

ソースコード 4: ソケットの作成 (サーバー側)

```

1  s_s=socket(AF_INET, SOCK_STREAM, 0);
2  if(s_s==-1){
3      fprintf(stderr,"socket\n");
4      return -1;
5  }

```

ここで TCP/IP 用の sockaddr 構造体のメンバを設定する。メンバの値は以下のように設定した。

ソースコード 5: sockaddr 構造体のメンバ設定

```

1  sa.sin_family = AF_INET;
2  sa.sin_addr.s_addr = htonl(INADDR_ANY);
3  sa.sin_port = htons((uint)atoi("61002"));

```

sockaddr 構造体 sa のメンバの設定について説明する．1 行目では IP プロトコルを IPv4 に，2 行目では IP アドレスによる接続を制限に関する設定をしている．上記の設定では，IP アドレスによる制限を設けていない．3 行目は待ち受けポート番号を設定している．

ソケットの待ち受け設定とソケットでサーバーとしての待ち受け設定を開始する．待ち受け設定は bind 関数，待ち受け設定の開始は listen 関数を用いる．以下に 2 つの関数を使用したプログラムを示す．

ソースコード 6: ソケットの待ち受け設定とソケットでサーバーとしての待ち受け設定を開始

```
1  if(bind(s_s, (struct sockaddr*)&sa, sizeof(sa)) == -1)
2  {
3      fprintf(stderr, "bindERROR (%s)\n", strerror(errno));
4      return -1;
5  }
6  printf("bind is done\n");
7
8  if(listen(s_s, 5) == -1)
9  {
10     fprintf(stderr, "listenERROR\n");
11     return -1;
12 }
13 printf("listen is done\n");
```

1 行目ではソケットの待ち受け設定を bind 関数で行っている．第 1 引数に待ち受け用ソケットを渡し，設定として第 2 引数に sockaddr 構造体を与えている．sockaddr の終わりがわかるように，sa のサイズを渡している．bind 関数のエラー処理は 1 から 5 行目で行っている．8 行目では待ち受け設定の開始を listen 関数で行っている．サーバーとして待ち受けを開始したソケットを第 1 引数に渡し，一度に通信できる相手の最大数を第 2 引数で指定している．本プログラムでは 5 人と最大通信できる．

この後の処理は，クライアントと連続して通信をするために繰り返し処理を行う．初めに，listen 状態中にソケットへきた接続要求を受け取り，データ通信用のソケットを作成する．

ソースコード 7: データ通信用のソケットを作成

```
1  len = sizeof(ca);
2  news = accept(s_s, (struct sockaddr*)&ca, &len);
3  if(news == -1){
4      err_num = errno;
5      fprintf(stderr, "acceptingERROR (%s)\n", strerror(err_num));
6      return 0;
7  }
8  printf("Connection accepted from %s:%d\n", inet_ntoa(ca.sin_addr), ntohs(ca.sin_port));
```

2 行目の accept 関数でデータ通信用のソケットを作成している．accept 関数の返り値が通信用のソケットとなる．本プログラムでは，通信用のソケットは news とした．また第 2 引数は接続相手に関する情報を取得するため sockaddr 構造体 ca を渡した．次の処理はクライアントからのメッ

表 1: プログラムの使用方法

コマンド	動作
%Q	終了
%C	登録件数の表示
%P n	先頭から n 件表示
%R file	file から読み込み
%W file	file へ書き出し
%F word	word を検索
%S n	データを n 番目の項数で整列

セージの受信と，それに対する応答の送信である．以下にプログラムを示す．

ソースコード 8: クライアントからのメッセージの受信と，それに対する応答の送信

```

1  if(recv(news,buf,BUF_SIZE,0) == -1){
2      fprintf(stderr,"recvERROR\n");
3  }
4  myprint(buf);
5  printf("receive is done\n");
6  if(strcmp(buf,"NON")!=0)isParse=parse_input(buf);
7
8  if(send(news,sending,BUF_SIZE,0)==-1)
9  {
10     fprintf(stderr,"sendERROR (%s)\n",strerror(errno));
11 }
12 printf("sending is done\n");
13 printf("~~~~~ALL DONE~~~~~\n");
14 close(news);

```

1 行目の `recv` 関数で通信相手から送信されてくるメッセージを `buf` に格納する．1 3 行目は `recv` 関数に関するエラー処理である．6 行目で `strcmp` 関数で受信メッセージが `NON` でない場合，`parse_input` 関数に `buf` を渡し，名簿プログラムを起動する．名簿管理プログラムでの出力結果はグローバル変数 `sennding` に格納されている．通信用ソケットを `send` 関数に渡し，名簿管理プログラムの出力結果を格納した `sending` をクライアントに送る．その処理が 8 行目で，以降 11 行目までがエラー処理である．受信と送信が完了したら，通信用ソケットを破棄する．サーバー側はクライアントからの通信を待ち続けるため，再びクライアントからの通信要求を `accept` 関数で待ち一連の処理を繰り返す．

3 プログラムの使用方法と使用例

本章ではプログラムの使用方法と使用例を述べる．表 1 に使用方法を示す．表 1 の主に使うコマンド (%Q, %C, %P, %R) 動作例を節に分けて示す．

3.1 サーバーとクライアントの起動

始めにサーバーとクライアントの起動について説明する．初めにサーバーを起動したときの動作を図 1 に，クライアントを起動したときの動作を図 2 に示す．

```
~/JikkenC/network/server$ ./my_program  
bind is done  
listen is done
```

図 1: サーバーを起動したときの動作

```
~/JikkenC/network/client$ ./my_program localhost index.html 61002  
connection is done
```

図 2: クライアントを起動したときの動作

サーバーではプログラムを開始するのみである．図 1 より，2.2 章で説明した bind と listen 関数の処理が完了している．対して，クライアントを開始する場合はコマンドライン引数の第 1 引数に接続先 IP アドレス (今回 localhost)，第 3 引数に接続先ポート番号を指定する．クライアントを開始した時点で，既にサーバーは接続要求を待っているため，通信が開始する．通信が開始したときのサーバーの動作を図 3 に示す．

```
bind is done  
listen is done  
Connection accepted from 127.0.0.1:42976
```

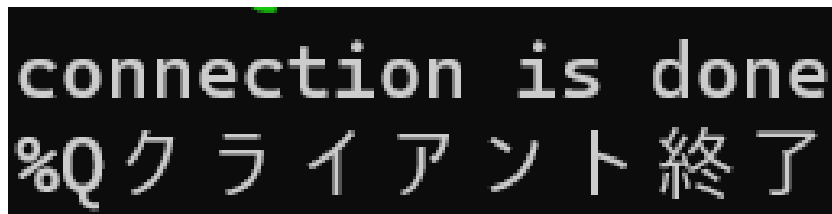
図 3: 通信が開始したときのサーバーの動作

図 2 と 3 の状態を確認したら，表 1 記載のコマンドをクライアント側にて入力する．

3.2 %Q の動作

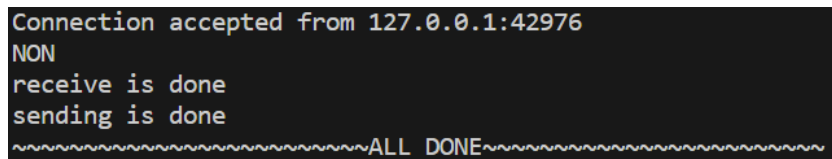
クライアントで %Q を入力したときの動作を図 4，サーバーで %Q を入力したときの動作を図 5 に示す．

図 4 より，%Q をクライアントで入力すると，クライアントプログラムが終了していることがわかる．次に図 5 より，%Q をクライアントから受け取ると，受け取ったバッファが”NON”になっており，空の文字列を送信し通信ソケットを破棄している．



```
connection is done
%Q クライアント終了
```

図 4: クライアントで %Q を入力したときの動作

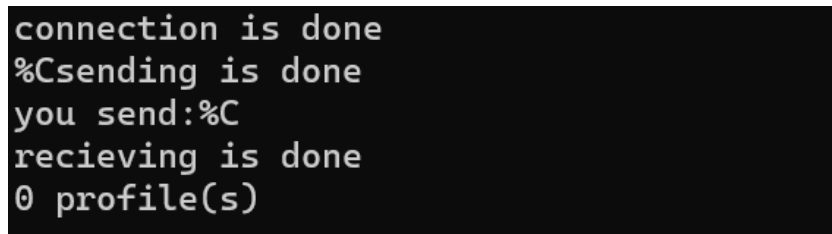


```
Connection accepted from 127.0.0.1:42976
NON
receive is done
sending is done
~~~~~ALL DONE~~~~~
```

図 5: サーバーで %Q を入力したときの動作

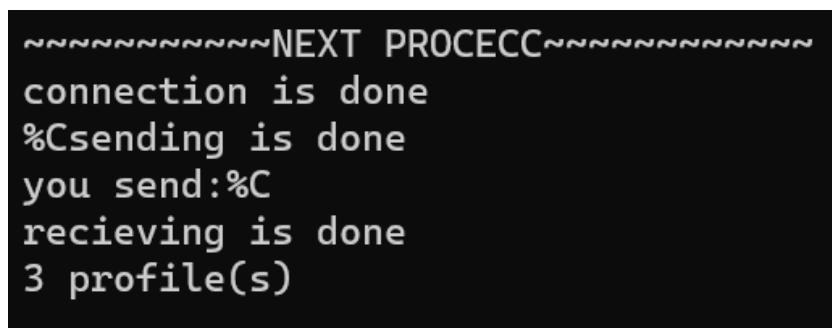
3.3 %C の動作

%C は登録件数を表示するコマンドである。プロフィールを登録していない状態で %C を入力したときの動作を図 6、3 件登録してから %C を入力したときの動作を図 7 に示す。



```
connection is done
%C sending is done
you send:%C
recieving is done
0 profile(s)
```

図 6: プロフィールを登録していない状態で %C を入力したときの動作



```
~~~~~NEXT PROCECC~~~~~
connection is done
%C sending is done
you send:%C
recieving is done
3 profile(s)
```

図 7: 3 件登録してから %C を入力したときの動作

何も登録していない状態で %C を入力すると 0 profiles と表示され、何も登録されていないこと

がわかる。対して、3 件登録した後に %C を入力すると 3 profiles と 3 件登録されていることがわかる。

3.4 %P

%P は登録してあるプロフィールを表示するコマンドである。%P 2 で 2 件プロフィールを表示したときの動作を図 8、%P 0 で登録してある全件を表示した時の動作を図 9 に示す。

```

XXXXXXXXXXNEXT PROCECCXXXXXXXXXX
connection is done
%P 2sending is done
you send:%P 2
recieving is done
Id      : 5101123
Name    : Reay Primary School
Birth   : 1925-03-01
Addr    : Reay Thurso
Com.    : 01847 811206 01847 811206 Primary 73 4.4 Open

Id      : 5101425
Name    : Mount Pleasant Primary School
Birth   : 1879-01-24
Addr    : Castletown Road Thurso
Com.    : 01847 893419 01847 892601 Primary 160 10.5 Integrated SEN Unit Open

```

図 8: 2 件プロフィールを表示したときの動作

```

XXXXXXXXXXNEXT PROCECCXXXXXXXXXX
connection is done
%P 0sending is done
you send:%P 0
recieving is done
Id      : 5101123
Name    : Reay Primary School
Birth   : 1925-03-01
Addr    : Reay Thurso
Com.    : 01847 811206 01847 811206 Primary 73 4.4 Open

Id      : 5101425
Name    : Mount Pleasant Primary School
Birth   : 1879-01-24
Addr    : Castletown Road Thurso
Com.    : 01847 893419 01847 892601 Primary 160 10.5 Integrated SEN Unit Open

Id      : 5101727
Name    : North Primary School
Birth   : 1868-11-05
Addr    : Girnigoe Street Wick
Com.    : 01955 602873 01955 602873 Primary 185 11.0 Open

Id      : 5107725
Name    : Bonar Bridge Primary School
Birth   : 1869-10-21
Addr    : Bonar Bridge Sutherland
Com.    : 01863 766221 1863766195 Primary 44 3.7 Open

```

図 9: 登録してある全件を表示した時の動作

前提として、コマンドを入力するタイミングで 4 件プロフィールを登録してある。図 8 より、%P 2 と入力すると 2 件分のプロフィールを受信していることがわかる。%P 0 は全件表示するコマン

ドで，動作例の図 9 より登録してあるプロフィール 4 件全て表示されていることがわかる．

3.5 %R

%R はサーバー側で svc 形式のファイルを読み込みするコマンドである．クライアント側でファイル読み込みする前のプロフィール件数と，読み込んだ後のプロフィール件数を表示し，動作を確認する．なお，読み込む前は 4 件プロフィールを登録しており，読み込むファイルには 121 件プロフィールが書き込まれている．読み込む前のプロフィール件数を図 10，ファイルを読み込んだ後のプロフィール件数を図 11 に示す．

```
~~~~~NEXT PROCECC~~~~~
connection is done
%Csending is done
you send:%C
recieving is done
4 profile(s)
```

図 10: 読み込む前のプロフィール件数

```
~~~~~NEXT PROCECC~~~~~
connection is done
%Csending is done
you send:%C
recieving is done
125 profile(s)
```

図 11: ファイルを読み込んだ後のプロフィール件数

図 10 では既に登録されてある 4 件のプロフィール件数が表示されている．対して，図 11 では 4 件のプロフィールに加えて，121 件新たにプロフィールを読み込んだことから 125 件プロフィールが登録されていることがわかる．%R コマンドは正しく動作しているといえる．

4 プログラムの作成過程に関する考察

%P 0 で全件を表示しようとするとき，100 から 200 程度のプロフィール件数であれば問題なく表示することができた．しかし 1000 件程度になるとバッファサイズが足りずサーバー側で

Segmentation Fault が発生した．この問題に対して，バッファサイズを 100 万にし，メモリに余裕を持たせることで解決した．しかし，常に 100 万バイトメモリを確保してあることは効率がわるいため，改善案として malloc 関数を用いて最低限のバッファサイズに加えて，件数表示に必要なバッファサイズを動的に確保することで負荷の少ないプログラムになる．

5 得られた結果に関する考察

コマンド %P で登録されているプロフィールを表示することができる．この機能を応用して，サーバー側から送られてきたプロフィールに関するバッファをクライアント側で標準出力することでサーバー側のファイルをクライアント側に転送する技術が実現できる．