情報工学実験C ネットワークプログラミング

氏名:大西　隼也
学籍番号:09427510

出題日: 2017 年 12 月 12 日
提出日:2018 年 1 月 30 日
締切日:2018 年 1 月 30 日

1　概要

本実験では，基本的な通信方式である TCP/IP，UDP/IP によるネットワークプログラミングについて学習する．また，分散システムの基本的な形式であるクライアントサーバモデルの仕組みを学習する．最終的に，クライアントサーバモデルに基づくプログラムを作成する．

2　クライアント・サーバモデルの通信の仕組みについて

クライアントサーバモデルとは，クライアントとサーバを分離して管理するソフトウェアモデルであり，今回の実験ではローカルの環境下で動作するプログラムを作成したが，ここでは，代表的なサーバモデルである，メールサーバや web サーバとクライアントがインターネットを通じて通信する際の方法について概要を説明した後，実装したプログラムのソケットを用いた通信について述べる．

2.1　インターネットでの通信の仕組み

インターネット上のすべての計算機には，一意の IP アドレスが割り振られているため，IP アドレスを用いて計算機を特定することができる．しかし実際に IP アドレスを用いて通信を行おうとすると，例えば岡山大学の IP アドレス (150.46.30.130) など数字の羅列で人間が直感的に分かりにくいため，インターネット上のホスト名（www.okayama-u.ac.jp）と IP アドレスを対応させるシステム，DNS(Domain Name Service) が用いられている．

DNS もサーバの一種であり，ターミナル上で nslookup コマンドなどを用いて処理を依頼すると，ホスト名から IP アドレスに（正引き），IP アドレスからホスト名に（逆引き）の変換結果を返す．

また同時に複数の計算機と通信する際や通信相手計算機に複数のプログラムが存在する場合には，IP アドレスに加えて補助アドレスとしてポート番号を利用する．

ポート番号とは，0-65535 の間で指定可能な数であり，サービスの種別を判断するために用いられる．例えば，IP アドレス（150.46.30.130）とポート番号（80）は岡山大学の web サーバを示す．

自作でプログラムを作成する際に注意したいのは，1023 番までのポートは well-known ポートと呼ばれ，主要なプロトコルで用いられる番号が決まっているため，それ以外の番号を利用する必要が有る．

- DNS(53)，HTTP(80)，POP3(110) など

2.2　今回作成したクライアントサーバモデルでの通信

以上述べてきたように，インターネットは TCP/IP という通信プロトコルを用いて通信を行っているが，実際に TCP/IP をプログラムから利用するには，プログラムとインターネットをつなぐ出入り口が必要になってくる．その出入り口となるのがソケットと呼ばれるものであり，TCP/IP 通信はソケット通信と呼ばれることもある．

ソケットの最大の特徴として，ソケットを介してデータを送受信する際の要領が基本的にファイル入出力と同じであり，扱いやすいという利点がある．そのため単純なプロセス間通信では，通信相手プロセスとの間にソケットを生成し，そのソケット番号を通信に利用しながらソケットに対して送信や受信の命令を実行することでデータの送受信を実現している．

3　名簿管理プログラムのクライアント・サーバプログラムの作成方針

3.1　名簿管理プログラムの仕様について

基本的にはプログラミング演習で作成した名簿管理プログラムの入出力部分を send 関数や recv 関数を用いて書き換えを行い，サーバ，クライアント間で通信を行えるように実装し直す．

今回与えられた仕様として，名簿管理プログラム終了時，クライアントのみ終了し，サーバ側のプログラムは接続待機状態に戻り待つというものがあったので，通信部を while 文で意図的に無限ループするような方針でプログラムを作成した．

4　プログラム及び，その説明

4.1　TCP/IP のプロトコルの説明

IP は，Internet Protocol の略称で，データグラムを転送するためのプロトコルとされている，アドレスとして IP アドレスとポート番号を用いる．

TCP は，Transmissiomn Control Protocol の略称で，ストリーム転送サービスを提供しており，これにより，信頼性と複数回に分けて送り出したデータについても，順序を保証することが可能になっている．しかし，その分通信に時間がかかるというデメリットもある．

4.2 名簿管理プログラムのコマンド一覧

コマンド	意味	備考
%Q	終了 (Quit)	
%C	登録件数などの表示 (Check)	
%P n	先頭から n 件表示 (Print)	n=0: 全件表示, n < 0 後ろから -n 件表示
%R file	file から読み込み (Read)	
%W file	file へ書き出し (Write)	
%F word	word を検索 (Find)	結果を%P と同じ形式で表示
%S n	データを n 番目の項目で整列 (Sort)	表示はしない
%D n	データを n 件削除 (Delete)	仕様は後述する
%A n	n 番目にデータを登録 (Add)	
%B	直前の状態に戻る (Back)	%R,%A の使用後のみ
%M	各コマンドの仕様を表示 (Manual)	

4.3 クライアントの処理の流れ

ここでは，クライアントプログラムの主な処理の流れと, 今回使用した TCP/IP の関数について述べる.

1. 通信相手の IP アドレスを取得
   - gethostbyname:IP アドレスを得る
2. ソケットの作成
   - socket:ソケットを作成する
3. 接続の確立
   - connect:コネクションを確立させる
4. 要求メッセージを送信
   - send:メッセージを送信する
5. 応答メッセージを受信
   - recv:メッセージを受信する
6. 応答メッセージを処理
7. ソケットの削除
   - close:ソケットを削除する

4.4 サーバの処理の流れ

サーバは要求メッセージの到着を常に待ち，要求メッセージが到着したら処理を行い，結果を送信する．ここでは，サーバプログラムの主な処理の流れと，今回使用した TCP/IP の関数について述べる．

1. ソケットの作成
    - socket:ソケットを作成する
2. ソケットに名前をつける
    - bind:ソケットに名前をつける
3. 接続要求の受付を開始する
    - listen:接続要求を待つ
4. 接続要求を受け付ける
    - accept:接続要求を受け付ける
5. 要求メッセージを受信
    - recv:メッセージを受信する
6. 要求メッセージを処理
7. 応答メッセージを送信する
    - send:メッセージを送信する
8. 次の接続要求の受付を開始する

5 プログラムの使用法

5.1 クライアントプログラムの動作

クライアント側のプログラムは起動後，ソケットを作成し，サーバ側との通信を確立した後，コマンドの入力待ちを行う．コマンドが入力されるとソケットを介してコマンドをサーバ側に送信し，処理結果を受け取り表示を行う．

```
oonishishunya-no-MacBook-Air:network oonishishunya$ ./meibomac-client localhost
クライアントの入力待ち
%C
登録件数は 0 件です.

クライアントの入力待ち
%R sample.csv
読み込みが完了しました．%C 等で確認してください.

クライアントの入力待ち
```

%C
登録件数は 2886 件です.

クライアントの入力待ち
%Q
終了します。

5.2 サーバプログラムの動作

サーバ側のプログラムは起動後，ソケットを作成し通信の準備が整ったらクライアントからの入力を待ち，%Q 以外のコマンドを受け取った際には処理を開始し結果をクライアントに返す.
%Q コマンドを受け取った際には，処理を行わず通信待機状態へと戻る.

```
oonishishunya-no-MacBook-Air:network oonishishunya$ ./meibomac-server
クライアントの入力待ち
入力 %C
サーバの処理開始
入力 after parse_line(): %C
処理終了

クライアントの入力待ち
入力 %R sample.csv
サーバの処理開始
入力 after parse_line(): %R sample.csv
処理終了

クライアントの入力待ち
入力 %C
サーバの処理開始
入力 after parse_line(): %C
処理終了

クライアントの入力待ち
入力 %Q
処理終了
```

6 プログラムの作成過程に関する考察

6.1 工夫した点

6.1.1 ソケットの再送待機状態対策

ソケットの特性上，一度ソケットをクローズすると，最初にクローズした側（今回の場合はクライアント側）は，再送待機状態 (TIME_WAIT) になる．この状態では，プログラム動作外で到着したパケットを破

棄できるよう，数分間は CLOSED 状態にならないので，他のソケットがそのポートを使用することができない．対策法としては，最も単純なのがプログラムを起動するたびに使用するポートを変えるなどが考えられるが，今回は作成したソケットに SO_REUSEADDR オプションを付加することで問題を解決した．

SO_REUSEADDR オプションを付加することで，同じローカルアドレスに bind を行ってもエラーにならず処理を行うことができる．

```
/* SO_REUSEADDR をつける*/

int ret;
ret = setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR,(const char *)&yes, sizeof(yes));
if(ret < 0){
  printf("Error : can't opt");
  return 0;
}
```

6.2 作成に苦労した点

6.2.1 %P コマンドの実装について

%P コマンドに代表されるように，サーバ側のプログラムから複数件の結果が返される際，パケットの分割サイズを予測することができないため，%P コマンドの出力件数が一つずれたり，send と recv のデータ送受信の関係からコマンド実行後，意図しない出力が行われコアダンプを起こすなどの問題が頻発した．

そこで，%P コマンドを入力した際には，実際にプリント処理を行う，recv，send のとは別に，何回プリント処理を行うか回数についてのみのデータの送受信を行うよう実装を行い，その後そのループ回数だけ recv を行うようクライアントプログラムを記述することで，問題を解決することができた．

```
    /*メッセージを受信する*/
    char kekka[MAX_LINE_LEN + 1];
    if((line[0]=='%' && line[1]=='P') || (line[0]=='%' && line[1]=='F')){
      bzero(&kekka, sizeof(kekka));
      recv(sockfd, kekka, sizeof(kekka), 0); //回数を受け取る
      int times;
      int l;
      times = atoi(kekka);
      for(l=0; l<times; l++){
bzero(&kekka, sizeof(kekka));
recv(sockfd, kekka, sizeof(kekka), 0);
printf("%s\n", kekka);
      }
```

7　得られた結果に関する考察

7.1　recv と send の対応付けに関する問題

今回の実験は，クライアントサーバモデルを構成して以前作成したプログラムを動かすというものだったので，実際の名簿管理プログラムの速度的な性能や機能面などの考察はテーマに沿わないため，6 章の作成過程に関する考察が中心となった．ここでは，プロセス間通信で想定外の動作を防ぐため特に注意し，実装に手間取った recv と send の対応付けに関する問題について考察する．

自作の名簿管理プログラムのコマンドの中で，recv と send の対応付けが一対一対応にならないものは，%P，%F，%A，%B の 4 つであった．

%P や%F の場合，問題になるのはプリント回数のみであるので，前述の考察で実装したように，プリント回数とプリント処理部をわけて通信を行うといった対応が可能だったが，%A や%B など自作で機能拡張したコマンドなどはその関数の実装法などが一般的ではない場合もあるため，今回はクライアント側ですべて場合分けをしてコマンドごとに recv と send の回数を合わせにいったが，現実的ではないことがわかる．

インターネットやメールサーバなどユーザ数や同時アクセスの問題を考えると，個々の処理はその関数の中で完結するように実装を行い，ユーザインタフェースにあたる，クライアント部などのプログラムは単純に記述を行うことが，後々の仕様変更や保守の観点から見れば重要であると考えられる．

7.2　考察に付随した感想

自分の作成したプログラムでも 1 年前ともなると何を書いているか，関数がどう動いているかわからない部分が多く，コメントやレポートを丁寧に書いておくことの重要性を実感した．また，先ほど考察したように，理想的には，名簿管理プログラムの内部で処理を完結させるような構造にしたかったが，当時プログラムが動けばさえすればいいと思ってコードを書いていた部分も多くあり，今回のように後から変更が加えづらい点でとても苦労したので，今後は保守性も意識してプログラムを作成しようと感じた．

8　作成したプログラム

今回，作成したプログラムのソースコードについて，名簿管理の処理を行うメインのプログラムに加え，その通信部分を担うクライアント，サーバプログラムの 3 つに分かれており，非常に膨大なページ数となるため github へのリンクと，プログラム名を記載することで割愛する．

`https://github.com/Shunya-Onishi/network`

- 名簿管理プログラム本体:meibo-prog.c p8-
- 名簿管理プログラムクライアント:meibo-client.c p22-
- 名簿管理プログラムサーバ:meibo-server.c p26-

8.1 追記:ソースコード

8.1.1 meibo-prog.c

```
 1 /*[1]*/
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5 #include <sys/types.h>
 6 #include <sys/socket.h>
 7
 8
 9 #define MAX_LINE_LEN 1024
10 #define MAXSTR 69
11 #define MAXPRO 10000
12 #define MAX_ID_LEN 31
13 #define MAX_BIRTH_LEN 10
14
15
16 int back = 0;
17 int ditems;
18 int mark = 0;
19 int flag =0;
20
21 /*[2]*/
22 struct date {
23   int y;
24   int m;
25   int d;
26 };
27
28 /*[3]*/
29 struct profile {
30   int id;
31   char name[MAXSTR+1];
32   struct date birthday;
33   char home[MAXSTR+1];
34   char *comment;
35 };
36
37 /*[4]*/
38 struct profile profile_data_store[MAXPRO];
39 int profile_data_nitems = 0;
40
```

```
41 void parse_line(char *line, int new_s);
42
43 /*[5]*/
44 int subst(char *str, char c1, char c2)
45 {
46   int n = 0;
47
48   while (*str) {
49     if (*str == c1) {
50       *str = c2;
51       n++;
52     }
53     str++;
54   }
55   return n;
56 }
57
58 /*[6]*/
59 int split(char *str, char *ret[], char sep, int max)
60 {
61   int cnt = 0;
62
63   ret[cnt++] = str;
64
65   while (*str && cnt < max) {
66     if (*str == sep){
67       *str = '\0';
68       ret[cnt++] = str + 1;
69     }
70     str++;
71   }
72  return cnt;
73 }
74
75 /*[7]*/
76 int get_line(FILE *fp,char *line)
77 {
78   if (fgets(line, MAX_LINE_LEN + 1, fp) == NULL)
79     return 0;
80
81   subst(line, '\n','\0');
82
83   return 1;
```

```
 84 }
 85
 86 /*[8]*/
 87 struct date *new_date(struct date *d, char *str)
 88 {
 89   char *ptr[3];
 90
 91   if (split(str, ptr, '-', 3) != 3)
 92     return NULL;
 93
 94   d->y = atoi(ptr[0]);
 95   d->m = atoi(ptr[1]);
 96   d->d = atoi(ptr[2]);
 97
 98   return d;
 99 }
100
101 /*[9]*/
102 struct profile *new_profile(struct profile *p, char *csv){
103   char *ptr[5];
104
105   if (split(csv, ptr, ',', 5) != 5)
106     return NULL;
107
108   p->id = atoi(ptr[0]);
109
110   strncpy(p->name, ptr[1], MAXSTR);
111   p->name[MAXSTR] = '\0';
112
113   if (new_date(&p->birthday, ptr[2]) == 0)
114     return 0;
115
116   strncpy(p->home, ptr[3], MAXSTR);
117   p->home[MAXSTR] = '\0';
118
119   p->comment = (char *)malloc(sizeof(char) * (strlen(ptr[4]) +1));
120   strcpy(p->comment, ptr[4]);
121
122   flag = 1;
123   return p;
124 }
125
126 /*[10]*/
```

```
127 void cmd_quit(char *param, int new_s)
128 {
129   char s[MAX_LINE_LEN + 1]={'\0'};
130   if(flag==0){
131     snprintf(s, MAX_LINE_LEN, "終了します。\n");
132     send(new_s, s, sizeof(s), 0);
133     // exit(0);
134       }
135   if(*param == 'a'){
136     snprintf(s, MAX_LINE_LEN, "終了します。\n");
137     send(new_s, s, sizeof(s), 0);
138     // exit(0);
139       }
140
141   else {
142     //    snprintf(s, MAX_LINE_LEN,"入力されたデータが保存されていません. \n%%Q a でこのまま終了します. \n");
143     snprintf(s, MAX_LINE_LEN, "終了します。\n");
144     send(new_s, s, sizeof(s), 0);
145   }
146 }
147 /*[11]*/
148 void cmd_check(int new_s)
149 {
150   //printf("登録件数は%d 件です. \n", profile_data_nitems);
151   char s[MAX_LINE_LEN + 1] = {'\0'};
152   snprintf(s, MAX_LINE_LEN, "登録件数は%d 件です. \n", profile_data_nitems);
153   send(new_s, s, sizeof(s), 0);
154 }
155 /*[12]*/
156 char *date_to_string(char buf[], struct date *date)
157 {
158   sprintf(buf, "%04d-%02d-%02d", date->y, date->m, date->d); /*文字列の中に入れる*/
159   return buf;
160 }
161
162 /*[13]*/
163 void print_profile(struct profile *p, int new_s)
164 {
165   char date[11];
166   char s[MAX_LINE_LEN+1] = {'\0'};
167
```

```
168   snprintf(s, MAX_LINE_LEN,"Id    : %d\nName  : %s\nBirth : %s\nAddr  : %s\nCom.  : %s\n",
169    p->id, p->name, date_to_string(date, &p->birthday), p->home, p->comment);
170   send(new_s, s, sizeof(s), 0);
171
172 }
173
174 /*[14]*/
175 void cmd_print(int nitems, int new_s)
176 {
177   int i, end = profile_data_nitems;
178   char s[MAX_LINE_LEN +1]={'\0'};
179
180   if(nitems == 0){
181     snprintf(s, MAX_LINE_LEN, "%d", end);
182     send(new_s, s, sizeof(s), 0);
183     for(i=0;i<end;i++){
184       print_profile(&profile_data_store[i], new_s);
185       //     printf("\n");
186   }
187   }else if(0 < nitems){
188     if(nitems > end) nitems = end;
189     snprintf(s, MAX_LINE_LEN, "%d", nitems);
190     send(new_s, s, sizeof(s), 0);
191     for(i=0;i<nitems;i++){
192       print_profile(&profile_data_store[i], new_s);
193       //     printf("\n");
194     }
195   }else if(nitems < 0){
196     end=end+nitems;
197     if(end< 0) end = 0;
198     snprintf(s, MAX_LINE_LEN, "%d", end);
199     send(new_s, s, sizeof(s), 0);
200     for(i=end;i < profile_data_nitems;i++){
201       print_profile(&profile_data_store[i], new_s);
202       //      printf("\n");
203     }
204   }
205 }
206
207 /*[15]*/
208 void cmd_read(char *filename, int new_s)
209 {
210   char buffer[MAX_LINE_LEN + 1];
```

```
211   int a,b;
212   FILE *fp;
213   char s[MAX_LINE_LEN + 1] = "\0";
214
215   a = profile_data_nitems;
216   fp = fopen(filename, "r");
217
218   if(fp == NULL) {
219     snprintf(s, MAX_LINE_LEN, "ファイルがありません, ファイル名を確認してください. \n");
220     send(new_s, s, sizeof(s), 0);
221     return;
222   }
223   while(get_line(fp ,buffer))
224     {
225     new_profile(&profile_data_store[profile_data_nitems], buffer);
226     profile_data_nitems++;
227     back = 1;
228     ditems = 1;
229     }
230   b = profile_data_nitems;
231   fclose(fp);
232
233   ditems = b - a;
234   back = 1;
235   snprintf(s, MAX_LINE_LEN, "読み込みが完了しました. %%C 等で確認してください. \n");
236   send(new_s, s, sizeof(s), 0);
237 }
238
239 /*[16]*/
240 void fprint_profile_csv(int i, FILE *fp)
241 {
242   fprintf(fp,"%d,", profile_data_store[i].id);
243   fprintf(fp,"%s,", profile_data_store[i].name);
244   fprintf(fp,"%04d-%02d-%02d,", profile_data_store[i].birthday.y
245   ,profile_data_store[i].birthday.m, profile_data_store[i].birthday.d);
246   fprintf(fp,"%s,", profile_data_store[i].home);
247   fprintf(fp,"%s\n", profile_data_store[i].comment);
248 }
249
250 /*[17]*/
251 void cmd_write(char *filename, int new_s)
252 {
253   int i;
```

```
254   FILE *fp;
255   char *file = "writefile.csv";
256   char s[MAX_LINE_LEN + 1] = "\0";
257
258   if(*filename == 0) fp = fopen(file,"w");
259   else fp = fopen(filename, "w");
260
261   for(i = 0; i < profile_data_nitems; i++){
262     fprint_profile_csv(i,fp);
263   }
264
265   fclose(fp);
266
267   flag = 0;
268
269   snprintf(s,MAX_LINE_LEN,"書き込みが完了しました．ファイルを確認してください．\n");
270   send(new_s, s, sizeof(s), 0);
271 }
272
273 /*[18]*/
274 void cmd_find(char *word, int new_s)
275 {
276   int i;
277   int count = 0;
278   struct profile *p;
279   char id[MAX_ID_LEN+1];
280   char date[MAX_BIRTH_LEN+1];
281   char s[MAX_LINE_LEN +1] = {'\0'};
282
283   for(i=0;i<profile_data_nitems;i++){
284     p = &profile_data_store[i];
285     sprintf(id, "%d", p->id);
286     if(strcmp(id, word) == 0 ||
287        strcmp(p->name, word) == 0 ||
288        strcmp(date_to_string(date, &(p->birthday)), word) == 0 ||
289        strcmp(p->home, word) == 0 ||
290        strcmp(p->comment, word) == 0
291        ){
292       count++;
293     }
294   }
295   snprintf(s, MAX_LINE_LEN, "%d", count);
296   send(new_s, s, sizeof(s), 0);
```

```
297
298   for(i=0;i<profile_data_nitems;i++){
299     p = &profile_data_store[i];
300     sprintf(id, "%d", p->id);
301     if(strcmp(id, word) == 0 ||
302        strcmp(p->name, word) == 0 ||
303        strcmp(date_to_string(date, &(p->birthday)), word) == 0 ||
304        strcmp(p->home, word) == 0 ||
305        strcmp(p->comment, word) == 0
306        ){
307       snprintf(s, MAX_LINE_LEN,
308        "Id    : %d\nName  : %s\nBirth : %s\nAddr  : %s\nCom.  : %s\n",
309        p->id, p->name, date_to_string(date, &p->birthday), p->home, p->comment);
310       send(new_s, s, sizeof(s), 0);
311     }
312   }
313 }
314
315 /*[19]*/
316 void swap(struct profile *a, struct profile *b)
317 {
318   struct profile tmp;
319
320   tmp = *a;
321   *a = *b;
322   *b = tmp;
323 }
324
325 /*[20]*/
326 int compare_date(struct date *d1, struct date *d2)
327 {
328   if (d1->y != d2->y) return d1->y - d2->y;
329   if (d1->m != d2->m) return d1->m - d2->m;
330   return d1->d - d2->d;
331 }
332
333 /*[21]*/
334 int profile_compare(struct profile *p1, struct profile *p2, int column)
335 {
336   switch (column){
337   case 1:
338     return p1->id - p2->id; break;
339   case 2:
```

```
  340     return strcmp(p1->name,p2->name); break;
  341   case 3:
  342     return compare_date(&(p1->birthday),&(p2->birthday)); break;
  343   case 4:
  344     return strcmp(p1->home,p2->home); break;
  345   case 5:
  346     return strcmp(p1->comment,p2->comment); break;
  347   }
  348   return 0;
  349 }
  350
  351
  352 /*[22]*/
  353 void cmd_sort(int param, int new_s)
  354 {
  355   int i, j;
  356   char s[MAX_LINE_LEN + 1] = {'\0'};
  357   struct profile *p;
  358
  359   if(0< param && param <6){
  360   for (i = 0; i < profile_data_nitems -1; i++) {
  361     for (j = 0; j < profile_data_nitems -1; j++) {
  362       p = &profile_data_store[j];
  363
  364       if (profile_compare(p, p+1, param) > 0)
  365 swap(p, p+1);
  366     }
  367   }
  368   back = 0;
  369   snprintf(s, MAX_LINE_LEN, "ソートが完了しました. %%P などで確認してください. \n");
  370   send(new_s, s, sizeof(s), 0);
  371   }else{
  372     snprintf(s, MAX_LINE_LEN, "有効な引数は 1~5 です. 正しい引数を入力してください.
\n");
  373     send(new_s, s, sizeof(s), 0);
  374   }
  375 }
  376
  377 /*[23]*/
  378 void ndelete(int nitems)
  379 {
  380   int i;
  381   for(i=0;i<nitems;i++){
```

```
382     free(profile_data_store[profile_data_nitems-1].comment);
383     profile_data_nitems--;
384   }
385 }
386
387 /*[24]*/
388 void cmd_delete(int param, int new_s)
389 {
390   int i;
391   FILE *fp;
392   char s[MAX_LINE_LEN + 1]={'\0'};
393   fp = fopen("backup.txt", "w");
394   mark = 0;
395
396   if(param == 0){
397     fprint_profile_csv(profile_data_nitems-1,fp);
398     ndelete(1);
399   }
400   else if(param > 0 && param< profile_data_nitems + 1){
401     for(i=0;i<param;i++)
402       fprint_profile_csv(profile_data_nitems-param+i,fp);
403     ndelete(param);
404   }
405   else if(param < 0 && -profile_data_nitems -1 <  param){
406     param = -param;
407     fprint_profile_csv(param-1,fp);
408     for(i=0;i<(profile_data_nitems -param);i++){
409       swap(&profile_data_store[param-1+i]
410    ,&profile_data_store[param+i]);
411     }
412     ndelete(1);
413     mark = param;
414   } else {
415     snprintf(s, MAX_LINE_LEN ,"保存件数は%d 件です\n 正しい引数を入力してください\n"
416     ,profile_data_nitems);
417     send(new_s, s, sizeof(s), 0);
418     return;
419   }
420   fclose(fp);
421   back = 2;
422   snprintf(s, MAX_LINE_LEN ,"delete ok\n");
423   send(new_s, s, sizeof(s), 0);
424 }
```

```
425
426 /*[25]*/
427 void cmd_add(int param, int new_s)
428 {
429   int i;
430   //  char line[MAX_LINE_LEN+1];
431   char s[MAX_LINE_LEN +1] = {'\0'};
432   struct profile *p;
433   mark = -param;
434
435   if(0 < param && param< profile_data_nitems){
436     snprintf(s, MAX_LINE_LEN, "CSV 形式で名簿データを入力してください. \n");
437     send(new_s, s, sizeof(s), 0);
438
439     //    get_line(stdin,line);
440     bzero(&s, sizeof(s));
441     recv(new_s, s, sizeof(s), 0);
442      parse_line(s, new_s);
443
444
445     p = &profile_data_store[profile_data_nitems-1];
446
447     for(i=0;i<(profile_data_nitems - param); i++){
448       swap(p-i-1,p-i);
449     }
450
451     back = 3;
452     /* snprintf(s, MAX_LINE_LEN, "登録が完了しました. \n"); */
453     /* send(new_s, s, sizeof(s), 0); */
454
455   }else if(param == profile_data_nitems){
456
457     snprintf(s, MAX_LINE_LEN, "CSV 形式で名簿データを入力してください. \n");
458     send(new_s, s, sizeof(s), 0);
459
460     //   get_line(stdin,line);
461     bzero(&s, sizeof(s));
462     recv(new_s, s, sizeof(s), 0);
463     parse_line(s, new_s);
464
465     p = &profile_data_store[profile_data_nitems-1];
466
467     swap(p-1,p);
```

```
468
469     back = 3;
470     /* snprintf(s, MAX_LINE_LEN, "登録が完了しました. \n"); */
471     /* send(new_s, s, sizeof(s), 0); */
472
473   }else{
474     snprintf(s, MAX_LINE_LEN, "登録件数は%d件です. 正しい引数を入力してください. \n",profile_data_ni
475     send(new_s, s, sizeof(s), 0);
476   }
477 }
478
479 /*[26]*/
480 void cmd_back(int new_s)
481 {
482   int i;
483   char s[MAX_LINE_LEN +1]={'\0'};
484   struct profile *p;
485
486   switch(back){
487
488   case 0:
489     snprintf(s, MAX_LINE_LEN,"%%B コマンドは, %%R,%%A コマンド実行後しか使用できませ
ん. \n");
490     send(new_s, s,sizeof(s), 0);
491     break;
492
493   case 1:
494     ndelete(ditems);
495     snprintf(s, MAX_LINE_LEN,"%%R コマンド実行前の状態に戻りました. \n");
496     send(new_s, s,sizeof(s), 0);
497     break;
498
499   case 2:
500     /* cmd_read("backup.txt", new_s); */
501     /* p = &profile_data_store[profile_data_nitems-1]; */
502     /* if(mark == 0){ */
503     /*   for(i=0;i<profile_data_nitems-mark;i++) */
504     /*  swap(p-1-i,p-i); */
505     /* } */
506     snprintf(s, MAX_LINE_LEN,"%%D コマンド実行前の状態に戻りました. \n");
507     send(new_s, s,sizeof(s), 0);
508     break;
509
```

```
510   case 3:
511     cmd_delete(mark, new_s);
512     // snprintf(s,MAX_LINE_LEN,"%%A コマンド実行前の状態に戻りました. \n");
513     // send(new_s, s,sizeof(s), 0);
514     break;
515
516   }
517
518   mark=0;
519   back=0;
520 }
521
522 /*[27]*/
523 void cmd_man(int new_s)
524 {
525   char s[MAX_LINE_LEN + 1] = {'\0'};
526   snprintf(s, MAX_LINE_LEN, "\n このプログラムは標準入力から「ID, 氏名, 年月日, 住所,
備考」からなるコンマ区切り形式 (CSV 形式) の名簿データを受け付けて, それらを名簿中に登録する名簿管
理プログラムである. \n 下記では, %%で始まる各コマンド入力の仕様を説明している. \n\n%%Q       |終了
(Quit)\n%%C       |登録件数などの表示 (Check)\n%%P n     |先頭から n 件表示 (Print)\n%%R file |file
から読み込み (Read)\n%%W file |file へ書き出し (Write)\n%%F word |word を検索 (Find)\n%%S n     |
データを n 番目の項目で整列 (Sort)\n%%D n     |データを n 件削除 (Delete)\n%%A n     |n 番目にデー
タを登録 (Add)\n%%B       |直前の状態に戻る (Back)\n%%M       |各コマンドの仕様 (Manual)\n\n");
527   send(new_s, s, sizeof(s), 0);
528 }
529
530 /*[28]*/
531 void exec_command(char cmd, char *param, int new_s) //全てのコマンドに new_s わたす
  いまは c だけ
532 {
533   char s[MAX_LINE_LEN + 1] = {'\0'};
534   switch (cmd) {
535   case 'Q': cmd_quit(param, new_s); break; //-----------------[10] -a コマンド以外
ok
536   case 'C': cmd_check(new_s); break; //-----------------------[11] ok
537   case 'P': cmd_print(atoi(param), new_s); break; //----------[14] ok
538   case 'R': cmd_read(param, new_s); break; //-----------------[15] ok
539   case 'W': cmd_write(param, new_s); break; //----------------[17] ok
540   case 'F': cmd_find(param, new_s); break; //-----------------[18] ok
541   case 'S': cmd_sort(atoi(param), new_s); break; //-----------[22] ok
542   case 'D': cmd_delete(atoi(param), new_s); break; //---------[24] ok
543   case 'A': cmd_add(atoi(param), new_s); break; //------------[25] ok
544   case 'B': cmd_back(new_s); break; //------------------------[26] case2(D) 以外 ok
```

```
545   case 'M': cmd_man(new_s); break; //-------------------------[27] ok
546   default:
547     snprintf(s, MAX_LINE_LEN, "%c は登録されていないコマンドです．%%M などでコマンド
を確認してください．\n",  cmd);
548     send(new_s, s, sizeof(s), 0);
549     break;
550   }
551 }
552
553
554
555 /*[29]*/
556 void parse_line(char *line, int new_s)
557 {
558   char s[MAX_LINE_LEN +1] = {'\0'};
559   if(line[0] == '%') {
560     exec_command(line[1], &line[3], new_s);
561   } else {
562     new_profile(&profile_data_store[profile_data_nitems], line);
563     profile_data_nitems++;
564     back = 1;
565     ditems = 1;
566     snprintf(s,MAX_LINE_LEN, "New Data added\n");
567     send(new_s, s, sizeof(s), 0);
568   }/* else if (new_profile(&profile_data_store[profile_data_nitems], line)!=NULL){
569       profile_data_nitems++;
570       back = 1;
571       ditems = 1;
572
573       send(new_s, s, sizeof(s), 0);
574       } else {
575       fprintf(stderr,"入力形式が違います．\n");
576       }*/
577 }
```

8.1.2 meibo-client.c

```
 1 #include <sys/fcntl.h>
 2 #include <sys/types.h>
 3 #include <sys/socket.h>
 4 #include <sys/stat.h>
 5 #include <netinet/in.h>
 6 #include <netdb.h>
 7 #include <stdio.h>
 8 #include <string.h>
 9 #include <strings.h>
10 #include <stdlib.h>
11 #include <unistd.h> //close
12 #include <fcntl.h>
13
14 #define PORT_NO 10016
15 #define MAX_LINE_LEN 1024
16
17 int subst(char *str, char c1, char c2)
18 {
19   int n = 0;
20
21   while (*str) {
22     if (*str == c1) {
23       *str = c2;
24       n++;
25     }
26     str++;
27   }
28   return n;
29 }
30
31 int get_line(FILE *fp,char *line)
32 {
33   if (fgets(line, MAX_LINE_LEN + 1, fp) == NULL)
34     return 0;
35
36   subst(line, '\n','\0');
37
38   return 1;
39 }
40
41 int main(int argc, char *argv[]){
```

```
42
43  /*通信相手の IP アドレスの取得*/
44
45  struct hostent* hostname;
46  if(argv < 0){
47    printf("Error : arguments number\n");
48  }
49
50  hostname = gethostbyname(argv[1]);
51  if(hostname == NULL){
52    printf("Error : hostname is NULL\n");
53  }
54
55  /*ソケットを作成する*/
56
57  int sockfd;
58  sockfd = socket(AF_INET, SOCK_STREAM, 0);
59  if(sockfd < 0){
60    printf("Error : can't make socket\n");
61    return(-1);
62  }
63
64  /*コネクションを確立する*/
65
66  struct sockaddr_in client_addr;
67
68  // memset((char*)&client_addr.sin_addr, 0, sizeof(client_addr.sin_addr));
69
70  memset((char*)&client_addr, 0, sizeof(client_addr));
71
72  client_addr.sin_family = hostname -> h_addrtype;
73  memcpy((char*)&client_addr.sin_addr, (char*)hostname -> h_addr, hostname -> h_length);
74  client_addr.sin_port = htons(PORT_NO);
75
76  if(connect(sockfd, (struct sockaddr *)&client_addr, sizeof(client_addr)) < 0){
77    printf("Error : can't connect\n");
78    return(-1);
79  }
80
81  /*メッセージを送信する*/
82
83  while(1){
84    printf("クライアントの入力待ち\n");
```

```
 85
 86     char line[MAX_LINE_LEN + 1];
 87     get_line(stdin, line);
 88     int check;
 89     check = send(sockfd, line, sizeof(line), 0);
 90
 91     if(check < 0){
 92       printf("Error : can't send\n");
 93       return(-1);
 94     };
 95
 96     /*メッセージを受信する*/
 97     char kekka[MAX_LINE_LEN + 1];
 98     if((line[0]=='%' && line[1]=='P') || (line[0]=='%' && line[1]=='F')){
 99       bzero(&kekka, sizeof(kekka));
100       recv(sockfd, kekka, sizeof(kekka), 0); //回数を受け取る
101       int times;
102       int l;
103       times = atoi(kekka);
104       for(l=0; l<times; l++){
105 bzero(&kekka, sizeof(kekka));
106 recv(sockfd, kekka, sizeof(kekka), 0);
107 printf("%s\n", kekka);
108       }
109
110     }else if(line[0]=='%' && line[1]=='A'){
111 bzero(&kekka, sizeof(kekka));
112 recv(sockfd, kekka, sizeof(kekka), 0);
113 printf("%s\n", kekka);
114
115 bzero(&kekka, sizeof(kekka));
116 get_line(stdin,kekka);
117 send(sockfd, kekka, sizeof(kekka), 0);
118
119 bzero(&kekka, sizeof(kekka));
120 recv(sockfd, kekka, sizeof(kekka), 0);
121 printf("%s\n", kekka);
122
123
124     }else if(line[0]=='%' && line[1]=='B'){
125        bzero(&kekka, sizeof(kekka));
126 recv(sockfd, kekka, sizeof(kekka), 0);
127 // printf("%s\n", kekka);
```

```
128 printf("Undo!\n");
129
130     }else if(line[0]=='%'){
131       bzero(&kekka, sizeof(kekka));
132       recv(sockfd, kekka, sizeof(kekka), 0);
133       printf("%s\n", kekka);
134       if(line[1]=='Q'){
135 //   exit(0);
136 return 0;
137       }
138     }else{
139       bzero(&kekka, sizeof(kekka));
140       recv(sockfd, kekka, sizeof(kekka), 0);
141       printf("%s\n", kekka);
142     }
143     bzero(&line, sizeof(line));
144     /* // while(1){ */
145     /*   char kekka[MAX_LINE_LEN + 1]; */
146     /*   if(recv(sockfd, kekka, sizeof(kekka), 0) < 0){ */
147     /*  printf("Error : can't recv\n"); */
148     /*  return(-1); */
149     /*   }; */
150
151     /*   printf("%s\n", kekka); */
152
153   }
154
155   /*ソケットを削除する*/
156   if(close(sockfd) < 0){
157     printf("Error : can't close\n");
158     return(-1);
159   }
160   return 0;
161 }
```

8.1.3 meibo-server.c

```
 1 #include <stdio.h>
 2 #include <string.h>
 3 #include <stdlib.h>
 4 #include <sys/fcntl.h>
 5 #include <sys/types.h>
 6 #include <sys/socket.h>
 7 #include <sys/stat.h>
 8 #include <netinet/in.h>
 9 #include <netdb.h>
10 #include <string.h>
11 #include <strings.h>
12 #include <unistd.h> //close
13 #include <fcntl.h>
14 #include <errno.h>
15
16 #define PORT_NO 10016
17 #define MAX_LINE_LEN 1024
18
19 void parse_line(char *line, int new_s);
20
21 int main(){
22
23   /*ソケットを作成する*/
24
25   int sockfd;
26   int yes = 1;
27   sockfd = socket(AF_INET, SOCK_STREAM, 0);
28   if(sockfd < 0){
29     printf("Error : can't make socket\n");
30     return(-1);
31   }
32
33   /* SO_REUSEADDR をつける*/
34
35   int ret;
36   ret = setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR,(const char *)&yes, sizeof(yes));
37   if(ret < 0){
38     printf("Error : can't opt");
39     return 0;
40   }
41
```

```
42  /*ソケットに名前をつける*/
43
44  struct sockaddr_in reader_addr;
45
46  memset((char*)&reader_addr, 0, sizeof(reader_addr));
47
48  reader_addr.sin_family = AF_INET; /*インターネットドメイン*/
49  reader_addr.sin_addr.s_addr = htonl(INADDR_ANY); /*任意の IP アドレスを受付*/
50  reader_addr.sin_port = htons(PORT_NO); /*接続待ちのポート番号を設定*/
51
52  int name;
53  name = bind(sockfd, (struct sockaddr *)&reader_addr, sizeof(reader_addr));
54  if(name < 0){
55    perror("bind");
56    printf("Error : bind\n");
57    return(-1);
58  }
59
60  /*接続要求を待つ*/
61
62  int wait;
63  wait = listen(sockfd,5);
64  if(wait < 0){
65    printf("Error : listen\n");
66    close(sockfd);
67    return(-1);
68  }
69
70  /*接続要求を受け付ける*/
71  int new_s;
72    while(1){
73
74  struct sockaddr_in client;
75
76  socklen_t len = sizeof(client);
77  new_s = accept(sockfd, (struct sockaddr *)&client, &len);
78  if(new_s < 0){
79    printf("Error : accept\n");
80    return(-1);
81  }
82
83  /*メッセージを受信する*/
84
```

```
 85   while(1){
 86     char buf[MAX_LINE_LEN + 1];
 87     printf("クライアントの入力待ち \n");
 88     recv(new_s, buf, sizeof(buf), 0);
 89     printf("入力 %s\n",buf);
 90     if(buf[0]=='%' && buf[1]=='Q'){
 91     parse_line(buf, new_s);
 92     printf("処理終了\n\n");
 93     break;
 94     }else{
 95       printf("サーバの処理開始 \n");
 96
 97       parse_line(buf, new_s);
 98
 99       printf("入力 after parse_line(): %s\n", buf);
100       bzero(&buf, sizeof(buf));
101       printf("処理終了\n\n");
102     }
103   }
104
105
106     /* /\*メッセージの送信*\/ */
107     /* memset(buf,0,sizeof(s)); */
108     /* send(new_s, buf, sizeof(buf), 0); */
109
110   }
111
112   /*ソケットの削除*/
113
114   close(new_s);
115
116   //  } dekai while
117
118   return 0;
119
120 }
```