情報ネットワーク実践論
第一回レポート

学籍番号：0312018310
氏名：馬場春樹

提出期限：2019/06/20
提出日：2019/06/20

1　ソースコード

Listing 1　”server_core.c”

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <arpa/inet.h>
4  #include <stdlib.h>
5  #include <unistd.h>
6  #include <fcntl.h>
7  #include <sys/file.h>
8  #include <errno.h>
9  #include <signal.h>
10 #include"errorCode.h"
11 #include"MessagePacket.h"
12 #include"userList.h"
13
14
15 #define MAX (3000)
16
17 int sock;        /* ソケットディスクリプタ*/
18 void IOSignalHandler(int signo); /* SIGIO 発生時のシグナルハンドラ*/
19 User* ulHead;
20
21 int main(int argc, char *argv[])
22 {
23   ulHead = mkList();
24   unsigned short servPort;    /* サーバローカル ()のポート番号 */
25   struct sockaddr_in servAddr;  /* サーバローカル ()用アドレス構造体 */
26   struct sigaction sigAction;  /* シグナルハンドラ設定用構造体*/
27
28   /* 引数の数を確認する. */
29   if (argc != 2) {
30     fprintf(stderr,"Usage:␣%s␣<Echo␣Port>\n", argv[0]);
31     exit(1);
32   }
33   servPort = atoi(argv[1]);
34
35   /* メッセージの送受信に使うソケットを作成する. */
36   sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
37   if (sock < 0) {
38     perror("socket()␣failed");
39     exit(1);
40   }
41   memset(&servAddr, 0, sizeof(servAddr));  /* 構造体をゼロで初期化*/
```

```
42    servAddr.sin_family = AF_INET;     /* インターネットアドレスファミリ*/
43    servAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* ワイルドカード*/
44    servAddr.sin_port = htons(servPort);    /* ローカルポート番号*/
45
46    if (bind(sock, (struct sockaddr *) &servAddr, sizeof(servAddr)) < 0) {
47      perror("bind()␣failed");
48      exit(1);
49    }
50
51    /* シグナルハンドラを設定する. */
52    sigAction.sa_handler = IOSignalHandler;
53
54    /* ハンドラ内でブロックするシグナルを設定する全てのシグナルをブロックする ().  */
55    if (sigfillset(&sigAction.sa_mask) < 0) {
56      perror("sigfillset()␣failed\n");
57      exit(1);
58    }
59    /* シグナルハンドラに関するオプション指定は無し. */
60    sigAction.sa_flags = 0;
61
62    /* シグナルハンドラ設定用構造体を使って, シグナルハンドラを登録する. */
63    if (sigaction(SIGIO, &sigAction, 0) < 0) {
64      perror("sigaction()␣failed\n");
65      exit(1);
66    }
67    /* このプロセスがソケットに関するシグナルを受け取るための設定を行う. */
68    if (fcntl(sock, F_SETOWN, getpid()) < 0) {
69      perror("Unable␣to␣set␣process␣owner␣to␣us\n");
70      exit(1);
71    }
72
73    /* ソケットに対してノンブロッキングと非同期 I/の設定を行う. O*/
74    if (fcntl(sock, F_SETFL, O_NONBLOCK | FASYNC) < 0) {
75      perror("Unable␣to␣put␣the␣sock␣into␣nonblocking/async␣mode\n");
76      exit(1);
77    }
78
79    /* メッセージの受信と送信以外の処理をする. */
80    for (;;) {
81      sleep(2);
82    }
83
84  }
85
86  /* SIGIO 発生時のシグナルハンドラ*/
87  void IOSignalHandler(int signo)
```

```
88 {
89   struct sockaddr_in clntAddr; /* クライアント用アドレス構造体*/
90   unsigned int clntAddrLen;  /* クライアント用アドレス構造体の長さ*/
91   char* msgBuffer=(char*)malloc(MAX);
92   memset(msgBuffer,'\0',MAX);
93   int recvMsgLen;     /* 受信メッセージの長さ*/
94   int sendMsgLen;     /* 送信メッセージの長さ*/
95   int recvContLen;
96   short recv_msgID,send_msgID;
97   char* contentBuffer=(char*)malloc(MAX);
98   memset(contentBuffer,'\0',MAX);
99   int uid;
100  char* sbuf=(char*)malloc(MAX);
101  memset(sbuf,'\0',MAX);
102  char* name=(char*)malloc(255);
103  memset(name,'\0',255);
104  int sl;
105  char* res=(char*)malloc(10);
106  memset(res,'\0',10);
107  char* msg=(char*)malloc(MAX);
108  memset(msg,'\0',MAX);
109  char* smsg=(char*)malloc(MAX);
110  memset(smsg,'\0',MAX);
111  int rclen;
112  User* ue=ulHead;
113  int sendl;
114  /* 受信データがなくなるまで，受信と送信を繰り返す．*/
115  do {
116    /* クライアント用アドレス構造体の長さを初期化する．*/
117    clntAddrLen = sizeof(clntAddr);
118
119    /* クライアントからメッセージを受信する．※この呼び出しはブロックしない() */
120    recvMsgLen = recvfrom(sock, msgBuffer, MAX, 0,
121      (struct sockaddr*)&clntAddr, &clntAddrLen);
122    /* 受信メッセージの長さを確認する．*/
123    if (recvMsgLen < 0) {
124      /* errono がEWOULDBLOCK である場合，受信データが無くなったことを示す．*/
125      /* EWOULDBLOCK は，許容できる唯一のエラー．*/
126      if (errno != EWOULDBLOCK) {
127        perror("recvfrom()␣failed\n");
128        exit(1);
129      }
130    } else {
131      /* クライアントのアドレスを表示する．IP*/
132      printf("Handling␣client␣%s\n", inet_ntoa(clntAddr. sin_addr));
133      recvContLen=Depacketize(msgBuffer,recvMsgLen,&recv_msgID,contentBuffer,MAX);
```

```
134       fprintf(stderr,"%d\n",recv_msgID);
135       switch(recv_msgID){
136         case MID_CHAT_TEXT:
137           sscanf(contentBuffer,"%d:%s",&uid,msg);
138           name=getNameByID(ulHead,uid);
139           sprintf(smsg,"%s:%s",name,msg);
140           rclen=Packetize(MID_CHAT_TEXT,smsg,MAX,sbuf,MAX);
141           while(ue->next!=NULL){
142             sendl=sendto(ue->next->socket, sbuf, rclen, 0,
143                     (struct sockaddr*)(ue->next->addr), sizeof(ue->next->addr));
144             fprintf(stderr,"sendl:%d\n",sendl);
145             ue=ue->next;
146           }
147           break;
148         case MID_JOIN_REQUEST:
149           strncpy(contentBuffer,name,strlen(name)+1);
150           uid=addUser(ulHead,sock,name,&clntAddr);
151           sprintf(res,"%d",uid);
152           rclen=Packetize(MID_JOIN_RESPONSE,res,strlen(res),sbuf,MAX);
153           fprintf(stderr,"%d:%d\n",rclen,strlen(res));
154           sendto(sock, sbuf, rclen, 0,
155             (struct sockaddr*)&clntAddr, sizeof(clntAddr));
156           free(name);
157           break;
158         case MID_LEAVE_REQUEST:
159           sscanf(contentBuffer,"%d",&uid);
160           delUser(ulHead,uid);
161           rclen=Packetize(MID_LEAVE_RESPONSE,"OK",strlen("OK"),sbuf,MAX);
162           sendto(sock, sbuf, rclen, 0,
163             (struct sockaddr*)&clntAddr, sizeof(clntAddr));
164           break;
165       }
166     }
167   } while (recvMsgLen >= 0);
168   free(sbuf);
169   free(msgBuffer);
170   free(msg);
171   free(smsg);
172 }
```

Listing 2 ”client.c”

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <arpa/inet.h>
4 #include <stdlib.h>
5 #include <unistd.h>
```

```
6  #include"errorCode.h"
7  #include"MessagePacket.h"
8  #define MAX 3000
9  #define TIMEOUT (2)     /* 関数のタイムアウト値 select 秒 [] */
10
11 /* キーボードからの文字列入力・サーバへの送信処理関数*/
12 int SendMessage(int sock, struct sockaddr_in *pServAddr);
13
14 /* ソケットからのメッセージ受信・表示処理関数*/
15 int ReceiveMessage(int sock, struct sockaddr_in *pServAddr);
16
17 int uid=-1;
18
19 int main(int argc, char *argv[])
20 {
21   char *servIP;      /* サーバのアドレス IP */
22   unsigned short servPort;  /* サーバのポート番号*/
23
24   int sock;       /* ソケットディスクリプタ*/
25   struct sockaddr_in servAddr; /* サーバ用アドレス構造体*/
26
27   int maxDescriptor;    /* 関数が扱うディスクリプタの最大値 select */
28   fd_set fdSet;      /* 関数が扱うディスクリプタの集合 select */
29   struct timeval tout;    /* 関数におけるタイムアウト設定用構造体 select */
30
31   /* 引数の数を確認する. */
32   if ((argc < 2) || (argc > 3)) {
33     fprintf(stderr,"Usage:␣%s␣<Server␣IP>␣[<Echo␣Port>]\n", argv[0]);
34     exit(1);
35   }
36
37   /* 第引数からサーバの 1アドレスを取得する. IP*/
38   servIP = argv[1];
39
40   /* 第引数からサーバのポート番号を取得する. 2*/
41   if (argc == 3) {
42     /* 引数が在ればサーバのポート番号として使用する. */
43     servPort = atoi(argv[2]);
44   }
45   else {
46     servPort = 9000;
47   }
48
49   /* メッセージの送受信に使うソケットを作成する. */
50   sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
51   if (sock < 0) {
```

```
52      fprintf(stderr, "socket()␣failed");
53      exit(1);
54    }
55
56    /* エコーサーバ用アドレス構造体へ必要な値を格納する．*/
57    memset(&servAddr, 0, sizeof(servAddr));    /* 構造体をゼロで初期化*/
58    servAddr.sin_family  = AF_INET;     /* インターネットアドレスファミリ*/
59    servAddr.sin_addr.s_addr = inet_addr(servIP); /* サーバのアドレス IP */
60    servAddr.sin_port    = htons(servPort);  /* サーバのポート番号*/
61
62    /* 関数で処理するディスクリプタの最大値として，ソケットの値を設定する．select*/
63    maxDescriptor = sock;
64
65    /* 文字列入力・メッセージ送信，およびメッセージ受信・表示処理ループ*/
66    for (;;) {
67
68      /* ディスクリプタの集合を初期化して，キーボートとソケットを設定する．*/
69      FD_ZERO(&fdSet);     /* ゼロクリア*/
70      FD_SET(STDIN_FILENO, &fdSet); /* キーボード標準入力 ()用ディスクリプタを設定 */
71      FD_SET(sock, &fdSet);    /* ソケットディスクリプタを設定*/
72
73      /* タイムアウト値を設定する．*/
74      tout.tv_sec = TIMEOUT; /* 秒*/
75      tout.tv_usec = 0; /* マイクロ秒*/
76
77      /* 各ディスクリプタに対する入力があるまでブロックする．*/
78      if (select(maxDescriptor + 1, &fdSet, NULL, NULL, &tout) == 0) {
79        /* タイムアウト*/
80        continue;
81      }
82
83      /* キーボードからの入力を確認する．*/
84      if (FD_ISSET(STDIN_FILENO, &fdSet)) {
85       /* キーボードからの入力があるので，文字列を読み込み，サーバへ送信する．*/
86        if (SendMessage(sock, &servAddr) < 0) {
87          break;
88        }
89      }
90
91      /* ソケットからの入力を確認する．*/
92      if (FD_ISSET(sock, &fdSet)) {
93        /* ソケットからの入力があるので，メッセージを受信し，表示する．*/
94        if (ReceiveMessage(sock, &servAddr) < 0) {
95          break;
96        }
97      }
```

```
98    }
99
100   /* ソケットを閉じ，プログラムを終了する．*/
101   close(sock);
102   exit(0);
103 }
104 /*
105  * キーボードからの文字列入力・サーバへのメッセージ送信処理関数
106  */
107 int SendMessage(int sock, struct sockaddr_in *pServAddr)
108 {
109   char string[MAX]; /* サーバへ送信する文字列*/
110   memset(string,'\0',MAX);
111   int stringLen;   /* サーバへ送信する文字列の長さ*/
112   int sendMsgLen;    /* 送信メッセージの長さ*/
113   char msg[MAX],pkt[MAX];
114   memset(msg,'\0',MAX);
115   memset(pkt,'\0',MAX);
116   const char *joinstr="!join";
117   const char *leavestr="!leave";
118   char ls[7];
119   char name[255];
120   char msgflg=0;
121   short msgid=MID_NONE;
122   short testid;
123   /* キーボードからの入力を読み込む．※改行コードも含まれる．() */
124   if (fgets(string, MAX, stdin) == NULL) {
125     /*「Control + 」が入力された．またはエラー発生．D*/
126     return -1;
127   }
128
129   /* 入力文字列の長さを確認する．*/
130   stringLen = strlen(string);
131   if (stringLen < 1) {
132     fprintf(stderr,"No␣input␣string.\n");
133     return -1;
134   }
135   if(stringLen>6){
136     strncpy(ls,string,5);
137     if(strcmp(ls,joinstr)==0){
138       sscanf(string,"!join␣%s\n",msg);
139       msgflg=1;
140       msgid=MID_JOIN_REQUEST;
141     }else{
142       strncpy(ls, string,6);
143       if(strcmp(ls,leavestr)==0){
```

```
144         if(uid==-1){
145           printf("please␣join\n");
146         }else{
147           sprintf(msg,"%d",uid);
148           msgflg=1;
149           msgid=MID_LEAVE_REQUEST;
150         }
151       }
152     }
153   }
154   if(!msgflg){
155     if(uid==-1){
156       printf("please␣join\n");
157     }else{
158       sprintf(msg,"%d:%s",uid,string);
159       msgid=MID_CHAT_TEXT;
160     }
161   }
162   sendMsgLen=Packetize(msgid,msg,(short)strlen(msg)+1,pkt,(int)sizeof(pkt));
163   memcpy(&testid,pkt,sizeof(short));
164   /* サーバへメッセージ入力された文字列 ()を送信する. */
165   sendMsgLen = sendto(sock, pkt, sendMsgLen, 0,
166     (struct sockaddr*)pServAddr, sizeof(*pServAddr));
167
168   return 0;
169 }
170
171 /*
172  * ソケットからのメッセージ受信・表示処理関数
173  */
174 int ReceiveMessage(int sock, struct sockaddr_in *pServAddr)
175 {
176   struct sockaddr_in fromAddr; /* メッセージ送信元用アドレス構造体*/
177   unsigned int fromAddrLen;  /* メッセージ送信元用アドレス構造体の長さ*/
178   char msgBuffer[MAX]; /* メッセージ送受信バッファ*/
179   memset(msgBuffer,'\0',MAX);
180   int recvMsgLen;     /* 受信メッセージの長さ*/
181   short msgid;
182   char contBuffer[MAX];
183   memset(contBuffer,'\0',MAX);
184   int contlen;
185   /* エコーメッセージ送信元用アドレス構造体の長さを初期化する. */
186   fromAddrLen = sizeof(fromAddr);
187
188   /* エコーメッセージを受信する. */
189   recvMsgLen = recvfrom(sock, msgBuffer, MAX, 0,
```

```
190     (struct sockaddr*)&fromAddr, &fromAddrLen);
191   if (recvMsgLen < 0) {
192     fprintf(stderr, "recvfrom()␣failed");
193     return -1;
194   }
195   contlen=Depacketize(msgBuffer,recvMsgLen,&msgid,contBuffer,MAX);
196   switch(msgid){
197     case MID_CHAT_TEXT:
198       printf("%s\n",contBuffer);
199       break;
200     case MID_JOIN_RESPONSE:
201       sscanf(contBuffer,"%d",&uid);
202       printf("join:id=%d\n",uid);
203       break;
204     case MID_LEAVE_RESPONSE:
205       printf("leaved\n");
206       uid=-1;
207   }
208   return 0;
209 }
```

Listing 3 ”MessagePacket.h”

```
1  /*-------------------------------------------------------------------------
2   *
3   * ■情報ネットワーク実践論（橋本担当分）課題共通ヘッダファイル3
4   *
5   * ［ファイル名］
6   * MessagePacket.h
7   *
8   * ［説明］
9   * このファイルには課題で利用するメッセージ3と，ID
10  * 送信パケット生成関数および受信メッセージ生成関数のプロトタイプ宣言が
11  * 含まれている．
12  *
13  * ［パケットのデータ構造］
14  * 1 2 3 4 5 6
15  * 0123456789012345 6789012345678901 2345678901234567 8901234567890123...
16  * +----------------+----------------+----------------+----------------
17  * | Message ID | Message Length | Message
18  * +----------------+----------------+----------------+----------------
19  *
20  * ・Message ID ：メッセージの種類を示すID バイト (2)
21  * ・Message ：メッセージの長さ Length バイト (2)
22  * ・Message ：メッセージ(Message Length で示されるバイト数)
23  * ※各メッセージに対応するメッセージのデータ構造は各自定義する．ID
24  *
```

```
25  */
26
27 #ifndef _MESSAGE_PACKET_H_
28 #define _MESSAGE_PACKET_H_
29
30 /*--------------------------------------------------------------------------
31  *
32  * ■メッセージ（※課題で必ず利用するメッセージ）ID3
33  *
34  */
35
36 /* メッセージ格納変数の初期化時などに利用するメッセージ IDID */
37 #define MID_NONE      (0x0000)
38
39 /* チャットテキスト*/
40 #define MID_CHAT_TEXT     (0x1010) /* クライアント<--> サーバ*/
41
42 /* グループへの参加要求と応答*/
43 #define MID_JOIN_REQUEST    (0x1021) /* クライアント---> サーバ*/
44 #define MID_JOIN_RESPONSE    (0x1022) /* クライアント<--- サーバ*/
45
46 /* グループからの退出要求と応答*/
47 #define MID_LEAVE_REQUEST    (0x1031) /* クライアント---> サーバ*/
48 #define MID_LEAVE_RESPONSE    (0x1032) /* クライアント<--- サーバ*/
49
50 /*--------------------------------------------------------------------------
51  *
52  * ■メッセージ（※課題の追加機能で利用するメッセージ）ID3
53  *
54  */
55
56 /* 特定の参加者に対するチャットテキスト*/
57 #define MID_PRIVATE_CHAT_TEXT  (0xa010) /* クライアント<--> サーバ*/
58
59 /* チャットグループ情報の要求と応答*/
60 #define MID_GROUP_INFO_REQUEST  (0xa021) /* クライアント---> サーバ*/
61 #define MID_GROUP_INFO_RESPONSE  (0xa022) /* クライアント<--- サーバ*/
62
63 /* チャットグループ内の参加者リスト要求と応答*/
64 #define MID_USER_LIST_REQUEST  (0xa031) /* クライアント---> サーバ*/
65 #define MID_USER_LIST_RESPONSE  (0xa032) /* クライアント<--- サーバ*/
66
67 /* 一定時間応答の無いクライアントに対しての接続確認要求と応答*/
68 #define MID_CONFIRMATION_REQUEST (0xa041) /* サーバ---> クライアント*/
69 #define MID_CONFIRMATION_RESPONSE (0xa042) /* サーバ<--- クライアント*/
70
```

```
/*--------------------------------------------------------------------------
 *
 * ■送信パケット生成関数（※課題で各自が必ず実装する関数）3
 *
 * ［関数名］
 * Packetize
 *
 * ［機能］
 * 送信メッセージを送信パケットバッファに格納する.
 * この関数呼び出しの後，send()/sendto() 関数を用いてパケットを送信する.
 *
 * ［引数］
 * ・msgID ：[IN ] メッセージ ID
 * ・msgBuf ：[IN ] メッセージバッファの先頭番地
 * ・msgLen ：[IN ] メッセージバッファに含まれるメッセージ長
 * ・pktBuf ：[OUT] 送信用パケットバッファの先頭番地
 * ・：pktBufSize[IN ] 送信用パケットバッファのサイズ
 *
 * ［戻り値］
 * ・送信パケットバッファ内のデータ長
 * ※エラー発生時はマイナスの値各自定義()を戻り値とする.
 *
 */
extern int Packetize(
        short msgID, char *msgBuf, short msgLen,
        char *pktBuf, int pktBufSize
);

/*--------------------------------------------------------------------------
 *
 * ■受信メッセージ生成関数（※課題で各自が必ず実装する関数）3
 *
 * ［関数名］
 * Depacketize
 *
 * ［機能］
 * 受信パケットバッファからメッセージを取得する.
 * recv()/recvfrom() 関数を用いてパケットをつ分受信した後に，1
 * この関数を呼び出してメッセージを取得する.
 *
 * ［引数］
 * ・pktBuf ：[IN ] 受信パケットバッファの先頭番地
 * ・pktLen ：[IN ] 受信パケットバッファに含まれる受信パケット長
 * ・msgID ：[OUT] メッセージ ID
 * ・msgBuf ：[OUT] メッセージバッファの先頭番地
 * ・：msgBufSize[IN ] メッセージバッファのサイズ
```

```
 *
 * ［戻り値］
 * ・メッセージバッファ内のメッセージ長
 * ※エラー発生時はマイナスの値各自定義()を戻り値とする.
 *
 */
extern int Depacketize(
        char *pktBuf, int pktLen,
        short *msgID, char *msgBuf, short msgBufSize
);

#endif /* _MESSAGE_PACKET_H_ */
```

Listing 4 ”MessagePacket.c”

```
#include"MessagePacket.h"
#include"errorCode.h"
#include<stdio.h>
#include<string.h>
int Packetize(short msgID, char* msgBuf, short msgLen, char* pktBuf, int pktBufSize)
{
        if ((sizeof(msgID)+sizeof(msgLen) + msgLen)>pktBufSize){
                return ILLEAGAL_MESSAGE_SIZE;
        }
        memset(pktBuf,'\0',pktBufSize);
        memcpy(pktBuf,&msgID,sizeof(msgID));
        memcpy(pktBuf+sizeof(msgID),&msgLen,sizeof(msgLen));
        memcpy(pktBuf+sizeof(msgID)+sizeof(msgLen),msgBuf,msgLen);
        return(sizeof(msgID)+sizeof(msgLen)+msgLen);
}

int Depacketize(char* pktBuf,int pktLen, short* msgID, char* msgBuf, short msgBufSize)
{
        short receivedMsgSize;
        memcpy(msgID,pktBuf,sizeof(short));
        memcpy(&receivedMsgSize,pktBuf+sizeof(short),sizeof(receivedMsgSize));
        if(receivedMsgSize>msgBufSize){
                return ILLEAGAL_MESSAGE_SIZE;
        }
        memset(msgBuf, '\0', msgBufSize);
        memcpy(msgBuf,pktBuf+sizeof(short)+sizeof(short),receivedMsgSize);
        return receivedMsgSize;
}
```

Listing 5 ”userList.h”

```
#ifndef _USER_LIST_H_
```

```
2 #define _USER_LIST_H_
3 #include<arpa/inet.h>
4 typedef struct User
5 {
6         struct User* prev;
7         struct User* next;
8         char* name;
9         int userID;
10        int socket;
11        struct sockaddr_in *addr;
12 }User;
13
14 int addUser(User* start,int socket,char *name,struct sockaddr_in *a);
15 int delUser(User* start,int userID);
16 int getSocketByID(User* start,int userID);
17 char* getNameByID(User* start,int userID);
18 User* mkList(void);
19
20 #endif
```

Listing 6 "userList.c"

```
1 #include"userList.h"
2 #include<stdlib.h>
3 #include<string.h>
4 int addUser(User* start,int socket, char* name,struct sockaddr_in *a){
5     User* cp=start;
6     while(cp->next!=NULL){
7         cp=cp->next;
8     }
9     User* c=(User*)malloc(sizeof(User));
10    c->next=NULL;
11    c->prev=cp;
12    cp->next=c;
13    c->userID=cp->userID+1;
14    c->socket=socket;
15    char* n=(char*)malloc(strlen(name)+1);
16    memset(n,0,strlen(name)+1);
17    memcpy(n,name,strlen(name));
18    cp->name=n;
19    struct sockaddr_in *addr=malloc(sizeof(struct sockaddr_in));
20    memcpy(addr,a,sizeof(struct sockaddr_in));
21    cp->addr=addr;
22    return c->userID;
23 }
24 int getSocketByID(User* start,int userID){
25    User* cp=start;
```

```
26      while(cp!=NULL){
27          if(cp->userID==userID){
28              return cp->socket;
29          }
30          cp=cp->next;
31      }
32      return 0;
33  }
34  char* getNameByID(User* start,int userID){
35      User* cp=start;
36      while(cp!=NULL){
37          if(cp->userID==userID){
38              return cp->name;
39          }
40          cp=cp->next;
41      }
42      return NULL;
43  }
44  int delUser(User* start,int userID){
45      User* cp=start;
46      while(cp->userID!=userID){
47          cp=cp->next;
48      }
49      cp->prev->next=cp->next;
50      cp->next->prev=cp->next;
51      free(cp->name);
52      free(cp->addr);
53      free(cp);
54      return 0;
55  }
56
57  User* mkList(void){
58      User* start=(User*)malloc(sizeof(User));
59      start->userID=0;
60      return start;
61  }
```

2 実行結果

Listing 7 ”サーバー側実行結果”

```
db-lab@dblab-ThinkPad-E490:~/final$ ./server 18310
Handling client 127.0.0.1
4129
5:1
Handling client 127.0.0.1
4129
5:1
Handling client 127.0.0.1
4112
Handling client 127.0.0.1
4112
```

Listing 8 ”クライアント側実行結果 1”

```
db-lab@dblab-ThinkPad-E490:~/final$ ./client 127.0.0.1 18310
!join lop
join:id=2
test
```

Listing 9 ”クライアント側実行結果 2”

```
db-lab@dblab-ThinkPad-E490:~/final$ ./client 127.0.0.1 18310
!join lala
join:id=1
ttt
```