

情報科学演習 C

第 2 回レポート

【担当教員】 内山 彰 教員

【提出者】 内藤 圭吾 (09B14049)
ソフトウェア科学コース・2 年
u434666c@ics.osaka-u.ac.jp

【提出日】 2015 年 1 月 28 日

1 クライアントの作成

1.1 プログラムの説明

まず、echo_server と同様にソケットを生成する。

```
if ( (sock=socket (AF_INET,SOCK_STREAM,IPPROTO_TCP)) <0) {  
    perror("socket");  
    exit(1);  
}
```

次に接続するサーバーの情報を構造体に格納し、そのサーバーに接続する。

```
bzero(&svr,sizeof(svr));  
svr.sin_family=AF_INET;  
svr.sin_port=htons(PORT);  
if ( ( cp = gethostbyname(argv[1]) ) == NULL ) {  
    fprintf(stderr,"unknown server %s\n",argv[1]);  
    exit(1);  
}  
bcopy(cp->h_addr,&svr.sin_addr,cp->h_length);  
  
connect(sock,&svr,sizeof(svr));
```

次に、標準入力から読み込み、パケットの送信を行う。標準入力から読み込む際、その文字列が EOF であるとき、つまり read 関数の戻り値が 0 であったときはプログラムを終了する。


```
bzero( buf , sizeof( buf ) );  
  
if ( read(0,buf,sizeof(buf)) == 0 ){  
    break ;  
}  
  
write(sock,buf,sizeof(buf));
```

最後にサーバーからメッセージを受信し、読み込み、標準出力へ書き込む。

```
if ( ( nbytes =read(sock,rbuf,sizeof(rbuf))) < 0)  
    perror("read");  
} else {  
    write(1,      rbuf,      sizeof(rbuf));  
}
```

この操作を while 文で繰り返すことで、繰り返しパケットを送信する。以上の方法でプログラムを実現した。

1.2 実行結果



The image shows two terminal windows side-by-side. The left window is titled 'exp184[5]%' and shows the command './echoclient exp205' being executed. The output is: 'Hello', 'Hello', 'hogehoge', 'hogehoge', 'bye', 'bye', 'closed'. The right window is titled 'exp205[3]%' and shows the command './echoserver' being executed. The output is: '[exp184.exp.ics.es.osaka-u.ac.jp]', 'closed', and an empty line.

```
exp184[5]% ./echoclient exp205
Hello
Hello
hogehoge
hogehoge
bye
bye
closed
exp184[6]% 

exp205[3]% ./echoserver
[exp184.exp.ics.es.osaka-u.ac.jp]
closed

```

1.3 考察

正しく送ったメッセージが帰ってきていることが分かる。また、EOFを読み込むとその時点で接続を切ってプログラムを終了しているの、正しく動作していることが分かる。

2 簡易 talk の作成

2.1 プログラムの説明

echo_client と同様にソケットを生成し、接続先の情報を設定し、接続する。

```
if ( (sock=socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0 ) {
    perror("socket");
    exit(1);
}

bzero(&svr, sizeof(svr));
svr.sin_family=AF_INET;
svr.sin_port=htons(PORT);
if ( ( cp = gethostbyname(argv[1]) ) == NULL ) {
    fprintf(stderr, "unknown server %s\n", argv[1]);
    exit(1);
}
bcopy(cp->h_addr, &svr.sin_addr, cp->h_length);

connect(sock, &svr, sizeof(svr));
```

サーバーと同様に select 関数を用いて標準入力とソケットを監視する。いずれかに入力があった場合の処理は echoclient と同様である。

```
if(select(sock+1, &rfd, NULL, NULL, &tv) > 0) {
    if(FD_ISSET(0, &rfd)) {
        read(0, buf, sizeof(buf));
        write(sock, buf, sizeof(buf));
        bzero(buf, sizeof(buf));
    }
```

```

    }
    if(FD_ISSET(sock,&rfd)) {
        read(sock,buf,sizeof(buf));
        write(0,buf,sizeof(buf));
        bzero(buf,sizeof(buf));
    }
}

```

2.2 実行結果

```

kterm
exp199[13]% ./simple-talk-client exp205
Please input your name!
Keigo
Thanks you!
HELLO
[naitou]:hogehoge
[naitou]:foo
Hello World
[naitou]:bye
□

kterm
exp205[15]% ./simple-talk-server
Please input your name!
Naitou
Thanks you!
[exp199.exp.ics.es.osaka-u.ac.jp]
[Keigo]:HELLO
Hogehoge
foo
[Keigo]:Hello World
bye
■

```

2.3 考察

発言順が交互以外であっても正しい順で表示され、受信した相手のメッセージの標準出力への反映するタイミングも正しいので、このプログラムが正しく動作していることが分かる。

3 拡張課題

3.1 拡張課題 1

3.1.1 実行結果

```

kterm
exp199[21]% ./echoclient exp205
Hello
hello
foo
foo
HOGEHOGE
hogehoge
□

kterm
exp205[23]% ./lowerechoserver
[exp199.exp.ics.es.osaka-u.ac.jp]
□

```

3.1.2 考察

このプログラムの実現には、echoserver が client からメッセージを受け取った際に、その各文字について、小文字へと変換する関数”tolower(char)”を実行し、その文字列を元の配列に格納することで実現することができた。このときに使用する関数を”tolower(char)”から”atoi(str)”や”atof(str)”等に変えることで、文字列を送信することで、それに対応する数値を返すプログラムを作ることも可能である。また、今回は関数を用いて実現したが、文字コードを操作することでも実現することができる。その場合、大文字に対して 32 を加えることで小文字へと変換することができる。

3.2 拡張課題 3

3.2.1 実行結果

実行結果は 2 章の実行結果に順ずる。

3.2.2 考察

メッセージを送信する際に、自身の名前を最初に付加したものを送ることで機能の実現をした。改善点としては、現在文字列を受信した際に、その文字列の発言者の名前を表示するだけで、こちら側が発言する際に、自身の名前が表示されてはいない。この改善方法としては、標準出力に文字列を出力した後、次の行に自身の名前を表示し、文字列の入力を待つ。その状態で、ソケットから文字列を受信したら、まず標準出力に”空白 + \r”を出力する。こうすることで行頭にカーソル位置を戻し、なおかつ現在標準出力に書き込まれている内容を削除する。空白の数はプログラムで想定している最大文字数 1024 文字にするのがよいと思われる。その後は、現在のプログラムと同様にソケットから文字列を読み込み標準出力に表示することで、実現できる。

3.3 拡張課題 4

3.3.1 実行結果

```
exp199[19]% ./simple-talk-client exp188
Please input your name!
Keigo
Thanks you!
[Naitou]:Hello
[Naitou]:Good Mornig
hogehoge
[Naitou]:foo
bye
[]

exp205[21]% ./simple-talk-client exp188
Please input your name!
Naitou
Thanks you!
Hello
Good Mornig
[Keigo]:hogehoge
foo
[Keigo]:bye
[]

exp188[50]% ./twoclientserver
client1 : [exp199.exp.ics.es.osaka-u.ac.jp]
client2 : [exp205.exp.ics.es.osaka-u.ac.jp]
[]
```

3.3.2 考察

プログラムの実現は、simple-talk-server では、受け取った文字列を標準出力に表示していたが、その部分を、もう一方のクライアントへの送信に書き換えることで実現した。今回は以下のように受け付けるクライアント数が2で実装した。

```
if ( ( csock = accept(sock, (struct sockaddr *)&clt, (socklen_t*)&clen) ) <0 ) {
    perror("accept");
    exit(2);
}

if ( ( csock2 = accept(sock, (struct sockaddr *)&clt2, (socklen_t*)&clen) ) <0 ) {
    perror("accept");
    exit(2);
}
```

しかし、この csock を増やしていくことで、3人以上のグループチャットの実装ができる。その際も、名前の表示を行うことで実用に耐えうるグループチャットの実装ができると考えられる。csock を増やした場合も select 関数の中の処理は csock、csock2 と同様のものでよいが、select 関数の第一引数は加えた csock の値を全て足し合わせる。

3.4 拡張課題5

3.4.1 実行結果



```
exp199[7]% ./udp_receive
hello from [192.168.16.188:53491]
hoge hoge from [192.168.16.188:163]
bye from [192.168.16.188:36682]
□

exp205[8]% ./udp_receive
hello from [192.168.16.188:53491]
hoge hoge from [192.168.16.188:163]
bye from [192.168.16.188:36682]
□

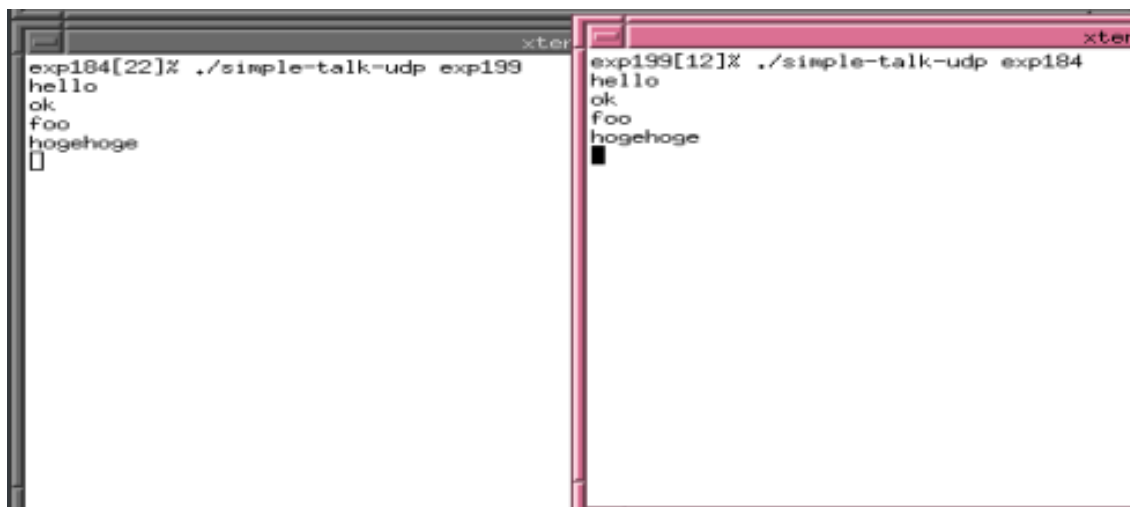
exp188[7]% ./udp_send hello
exp188[8]% ./udp_send hoge hoge
exp188[9]% ./udp_send bye
exp188[10]% █
```

3.4.2 考察

実現には、udp_send で送り先の ip アドレスを inet_aton 関数を用いて、”192.168.16.255”に指定し、さらに setsockopt 関数で SOL_SOCKET と SO_BROADCAST を設定することで実現した。今回はオプションとして SO_BROADCAST を設定したがオプションは他にもあり、それらを用いることでさまざまな付加機能をつけることができる。例えば、SO_SNDTIMEO オプションを用いると、サーバーがクライアントからの接続を待ち、一定時間接続が無かった場合、タイムアウトしてプログラムを終了する、といった機能をつけることができる。

3.5 拡張課題 6

3.5.1 実行結果



```
exp184[22]% ./simple-talk-udp exp199
hello
ok
foo
hogehoge
█

exp199[12]% ./simple-talk-udp exp184
hello
ok
foo
hogehoge
█
```

3.5.2 考察

今回はごく単純な最低限の仕様で実現したが TCP での簡易 talk と同様の付加機能 (名前の表示等) をつけるのは、同じ手法で実現できると考えられる。また、twoclientserver と同様の機能を udp_reseive の受け取ったときの処理を付け加えることで UDP でも実現できると考えられる。付け加える処理についても、twoclientserver と同様で、受け取った文字列を他方のソケットへ書き込むことで実装できると考えられる。

4 感想

通信に関するプログラムを作成したのは初めてだったので、使う関数から全て調べる必要があり大変ではあったが、とても勉強になりよかった。