第７話（チャットの仕組みⅡ）

タヌキ！チャットというのは、リアルタイムの文字の相互通信だということは知っているよな。相互通信をする為には、相互にプロセス（実行中のプログラム）を持っていなければならないことはわかるよな。お互いにスマホを持って会話するようなものだ。相互に持っているスマホは同じキャリア（販売店と考えて良い）の物でも、異なった物でも良いよね。でも仕組みは相互とも同じだよ。送信側と受信側に切り替わるだけです。それと同じようにチャットのプログラムもサーバプログラムとクライアントプログラムの作りは殆ど同じなんだ。それで、同じ説明を繰り返しても退屈するだけだから、説明は送信と受信の切り替えの部分だけしておくよ。OK！

OK、OK！同じことを繰り返し説明され、うんざりすることも多いからね。

では、サーバプログラムに問いかけるクライアントプログラム（cclient.c）を提示するよ。プログラムで読み込むヘッダファイル（プロが作ったプログラム）（netdb.h）には、通信に必要な変数の集合体（構造体）が定義されているから注意してね。

クライアントプログラム（cclient.c）---------1

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define PORT     (in_port_t)50000 /* サーバ(自分)のポート番号 */
#define BUF_LEN  512              /* 送受信のバッファの大きさ */

main()
{
        /* 変数宣言 */
        struct hostent  *server_ent; /* サーバ(相手)の情報 */
        struct sockaddr_in  server;  /* サーバ(相手)のアドレス */
        int soc;                     /* ソケットのディスクリプタ */
        char hostname[]="サーバ名";  /* サーバ(相手)のホスト名：localhost など */
                                     /* サーバのＩＰアドレスもＯＫ  */
        char buf[BUF_LEN];           /* 送受信のバッファ  */

        /* サーバ(相手)のホスト名からアドレス情報を得る  */
        if((server_ent = gethostbyname(hostname)) == NULL){
                perror("gethostbyname");
                exit(1);
        }

        /* サーバ(相手)のアドレスを sockaddr_in 構造体に格納  */
        memset((char *)&server, 0, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_port = htons(PORT);
        memcpy((char *)&server.sin_addr, server_ent->h_addr,
               server_ent->h_length);

        /* IPv4 でストリーム型のソケットを作成  */
        if((soc = socket(AF_INET, SOCK_STREAM, 0)) < 0){
                perror("socket");
                exit(1);
        }
```

クライアントプログラム（cclient.c）---------2

```
        /* サーバ(相手)に接続  */
        if(connect(soc, (struct sockaddr *)&server, sizeof(server)) == -1){
                perror("connect");
                exit(1);
        }

        /* 相手が先 */
        write(1, "Wait¥n", 5);

        /* 通信のループ */
        do{
                int n;                          /* 読み込まれたバイト数  */

                n = read(soc, buf, BUF_LEN); /* ソケット soc から読む   */
                write(1, buf, n);            /* 標準出力 1 に書き出す   */
                n = read(0, buf, BUF_LEN);   /* 標準入力 0 から読む     */
                write(soc, buf, n);          /* ソケット soc に書き出す */
        }
        while( strncmp(buf, "quit" ,4) != 0 );        /* 終了判定 */

        /* ソケットを閉じる */
        close(soc);
}
```

構造体　hostent　(netdb.h)

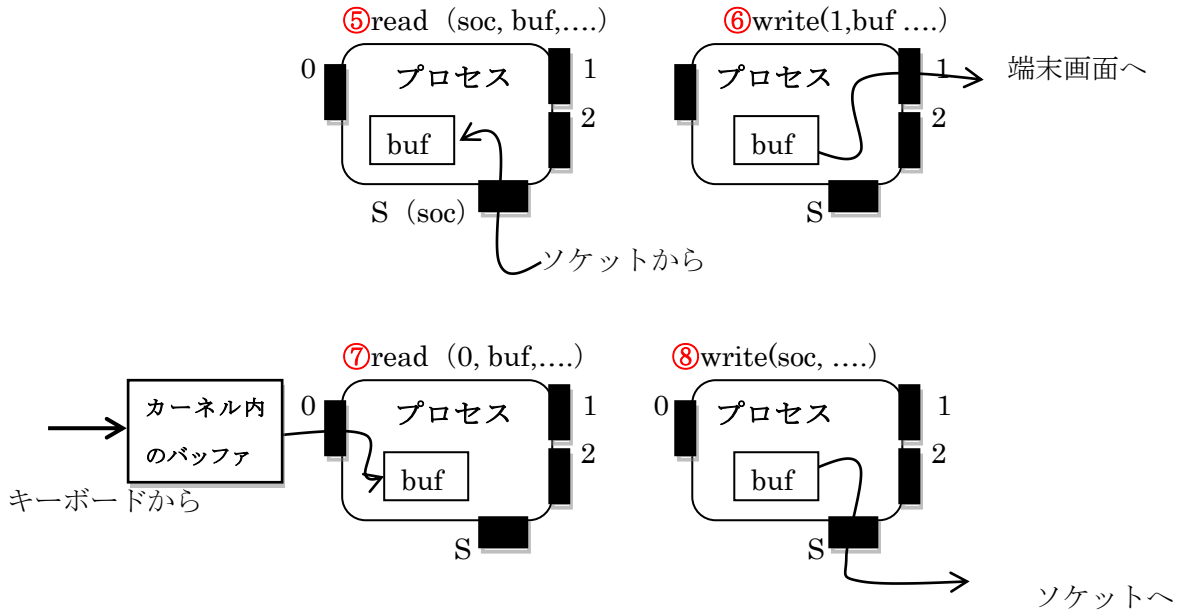
```
struct hostent {
    char  *h_name;            /* official name of host */
    char **h_aliases;         /* alias list */
    int    h_addrtype;        /* host address type */
    int    h_length;          /* length of address */
    char **h_addr_list;       /* list of addresses */
#define h_addr h_addr_list[0] /* h_addr は h_addr_list の最初のアドレス。*/
                              /*過去との互換性のため */
}
```

キツネ！構造体とか、サッパリわかんね・・・・！

そうだよな。C 言語で一番理解しにくい部分がポインタ変数（変数名にアスタリスク（*）が付いたもの）と構造体だからな。でも構造体とポインタ変数で疑似クラスが作れるんだ。つまり、オブジェクト指向言語のクラスの基本的考えに繋がっているので重要な部分でもあるんだが、C 言語をしっかり勉強した上でないと無理かもね。諦めるか・・・諦めたら動物界で生きていけないかもね。

C 言語がわからなくても、データ（メッセージ）の流れを図示するから、仕組みぐらいはわかるだろう。
⑤の read 命令でサーバからのメッセージがソケットを通ってクライアントプロセスのバッファ（buf）に入ってきます。⑥の write 命令でバッファ(buf)から指定のモニタにメッセージが表示されます。⑦の read 命令でクライアント側のキーボードから入力されたサーバ向けの返答メッセージがプロセスのバッファ（buf）に入力され、⑧の write 命令で割り当てられたファイルディスクリプタ(S)からソケットを通してサーバプロセスに運ばれます。
これは、タヌキでもわかるだろ。また、このようなプログラムもこのような仕組みを作ることのできるコンピュータも凄いと思わないか。

クライアントプロセス内のデータの流れ



キツネ！プログラムの３８行目にある connect()関数について教えてくれ！

OK！connect()関数は、クライアントプロセスがサーバプロセスと接続ができたかどうか確認する関数なんだ。つまり、上図の⑤と⑧のようにソケットディスクリプタ（S）が出入り口で、ソケットを通してプロセス間通信が確立できているか判断する関数だ。
接続でき、通信が可能な場合は関数の返り値として０を返すのさ。１の場合は、いくら頑張っても通信はできないよ。
詳細を次に書いておくよ！

connect()関数の詳細

ソケット（ディスクリプタ）を参照（相手先）している IP アドレスに接続する。接続が確立されたら、ソケットディスクリプタからの Read、Write が可能になる。

connect（ソケットディスクリプタ、送信先の IP アドレス、IP アドレスの長さ（４バイト））

connect(soc, (struct sockaddr *)&server, sizeof(server))

返り値：成功（確立）すれば 0、失敗すれば-1

プログラムの中に、C言語の関数がいくつか出てくるから、調べる時間を節約できるように解説しておくよ！

C言語の復習１　アロー演算子

server_ent->h_addr

受け取ったサーバの情報（ポインタ変数）をクライアントの構造体（hostent）のメンバー（h_addr）にコピーする。

server_ent：構造体 hostent のポインタ変数（ポインタ変数でなければならない）

->：アロー演算子

h_addr：構造体（hostent）のメンバー

C言語の復習２　メモリーの初期化

memset((char *)&server, 0, sizeof(server))

文字列にキャストした構造体 server（ポインタ変数）の始めから、構造体の長さまで、0で初期化している。

C言語の復習３　データのコピー

コピー先　　コピー元　　コピーするバイト数

memcpy((char *)&server.sin_addr, server_ent->h_addr,server_ent->h_length)

受け取ったサーバの IP アドレス（h_addr）をクライアントの構造体 sockaddr_in のメンバー（sin_addr）に h_length で指定されたバイト分コピーしている。

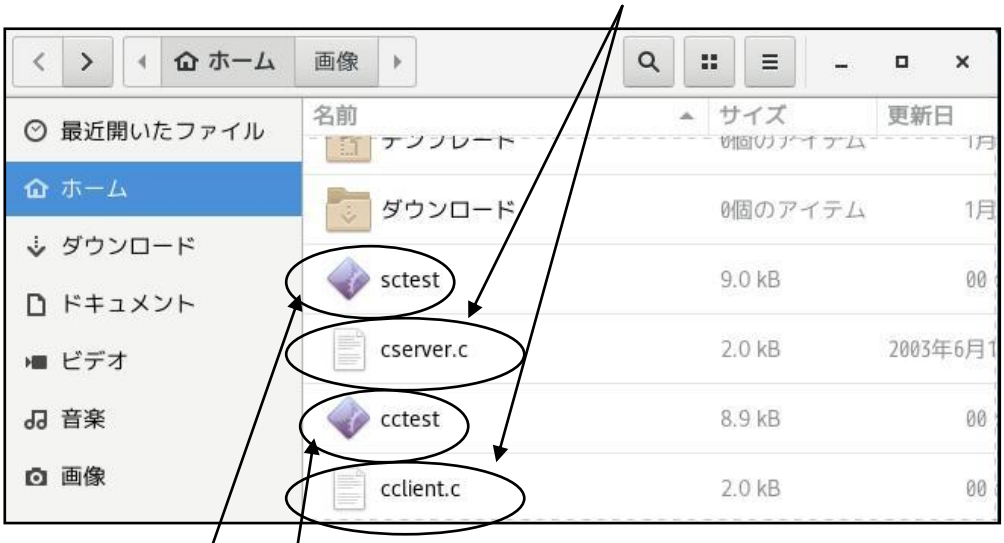
キツネ、プログラムを入力し、保存したけれど、その後どうすればいいんだ。

保存したサーバプログラムとクライアントプログラムをコンパイルし、実行し、２個のプロセスを１個のＯＳ（CentOS）上で実行するよ。２台の PC でチャットをしなければつまらないかもしれないが、動作確認するには１台の PC の OS 上で実行する方がプログラムのバグ取りが楽だよ。

次に、実行までの手順を示しておくよ。

チャットプログラムのコンパイル

①作成したソースファイル（cserver.c、cclient.c）を自分のホームディレクトリに保存します。



②コンパイルします。実行ファイルを各々「sctest」、「cctest」とします。

サーバの実行ファイルを作成

```
fox  @localhost~:$ gcc –o sctest cserver.c
```

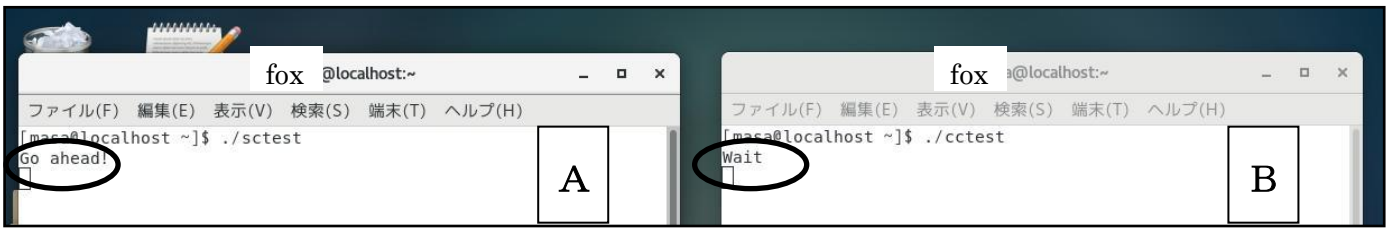
クライアントの実行ファイルを作成

```
fox  @localhost~:$ gcc –o cctest cclient.c
```

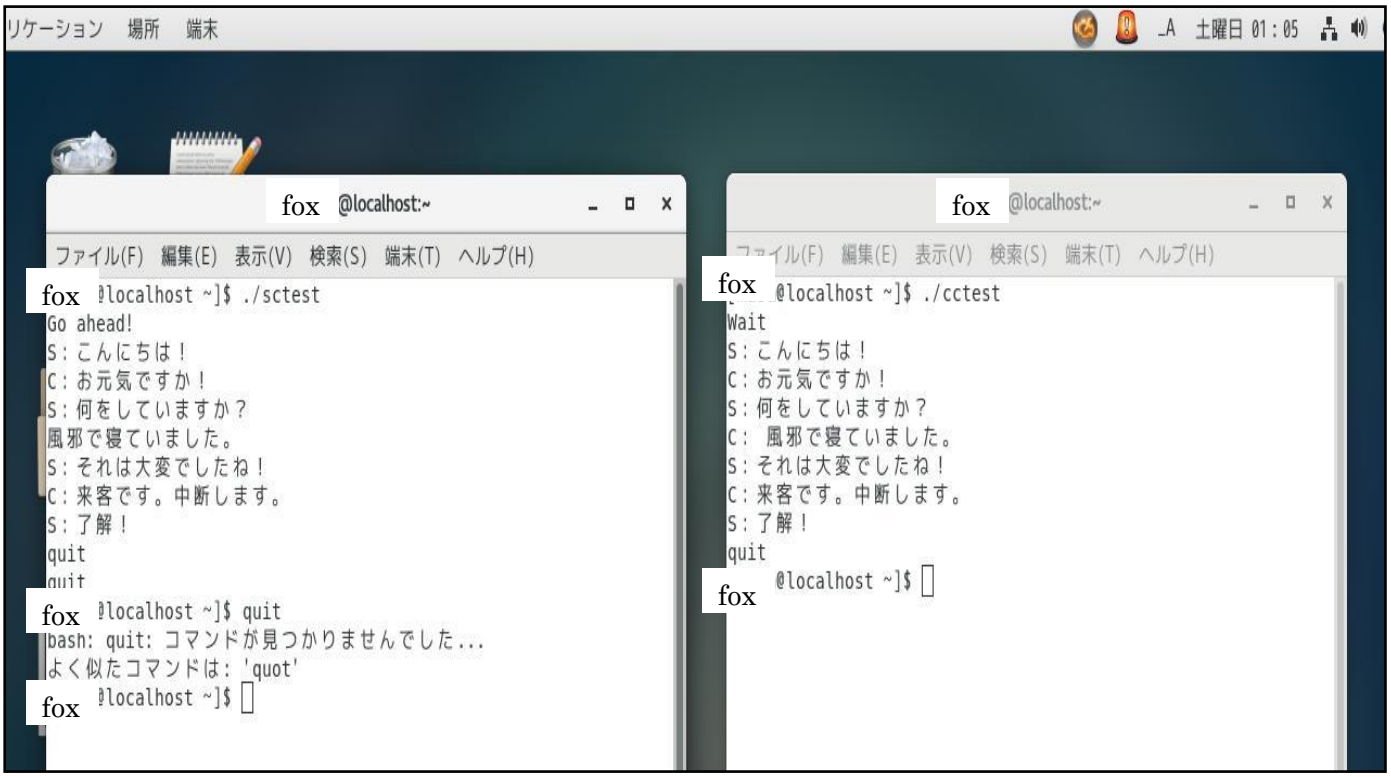
チャットプログラムの実行－１

①１台の PC（ホスト）でサーバプロセスとクライアントプロセスを実行する場合

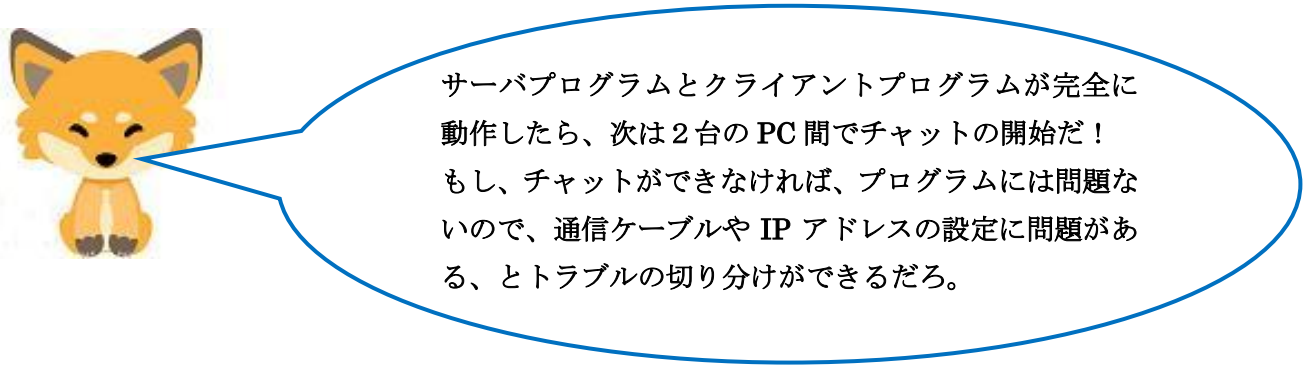
- 端末を２個起動します。サーバプロセスを起動する方を［A］、クライアントプロセスを起動する方を［B］とします。
- 先に［A］をアクティブにし、「./sctest」と入力し、待ちの状態にします。
- 次に［B］をアクティブにし、「./cctest」と入力します。次の瞬間、［A］に「Go ahead」（私が先だよ）、［B］に「Wait」（待つよ）というメッセージが表示されます。



②先に［A］、次に［B］という順番の繰り返しでチャットを始めます。手順を間違えると正しく応答しなくなります。

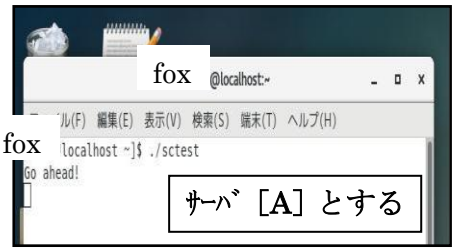


③終わらせる時は、「quit」と入力します。



チャットプログラムの実行－２

①２台以上の PC（ホスト）で１つのサーバプロセスと複数のクライアントプロセスを実行する場合



・サーバプロセスを起動するＰＣで端末を起動します。それをサーバ［A］とします。
・クライアントプロセスを起動する端末を起動します。それをクライアント［B］とします。
　［重要］起動前に、クライアントプログラムの修正とコンパイルが必要ですよ！

```
char hostname[]="接続を依頼するサーバのＩＰアドレスを設定";
```

・先に［A］をアクティブにし、「./sctest」と入力し、待ちの状態にします。
・次に［B］をアクティブにし、「./cctest」と入力します。次の瞬間、［A］に「Go ahead」（私が先だよ）、［B］に「Wait」（待つよ）というメッセージが表示されます。
・後は、チャットプログラムの実行－１と同じです。

おもろい、おもろい！
人の作ったプログラムをアプリとして実行するだけでなく、仕組みを理解した上でチャットをするのは達成感があるね。ここまでくるのは大変だったけれど、満足感があるよ。キツネ、ありがとう。

どういたしまして。俺もタヌキに喜んでもらえて嬉しいよ。

タヌキ！少し休憩し、

第８話に進むか！