2006 年度プログラミング演習３
クライアント／サーバ課題
補足資料：サーバプログラムの作成手順

ここでは，ソケット機能を使ったサーバプログラムの作成手順について説明する．サーバプログラムの処理手順は以下のようになる．

1. socket()によるソケット生成
2. bzero()による sockaddr_un 構造体の初期化
3. sockaddr_un 構造体の設定
4. unlink()によるソケットのアンリンク
5. bind()によるソケット登録
6. listen()によるソケットの接続準備
7. accept()によるクライアントからの接続待機
8. read() / write()によるクライアントとのデータ送受信.
9. close()によるソケットのクローズ.

以下ではステップ 4-1 のプロセス 2 のサンプルプログラムを基に上記の手順について解説を行う．

```
/*************************************************************/
/* Step4-1_Proc-2.c：同一計算機上でのメッセージの送受　プロセス２ */
/*************************************************************/

#include <stdio.h>  // fgets(), printf(), stdin()
#include <string.h> // strcpy(), strlen()
#include <sys/socket.h> // socket(), connect(), bind(), listen(), accept()
#include <sys/types.h> // socket(), connect(), bind(), accept()
#include <sys/un.h> // struct sockaddr_un
#include <unistd.h> // unlink()

#define EVER       1
#define MAXLENGTH 256

#define SOCKETNAME "/tmp/socket" // 通信に使うソケットファイル
                                 // クライアントと同じファイルを指定する必要がある
```

- **#define SOCKETNAME "/tmp/socket"**
  ソケットを用いた PF_UNIX（UNIX ドメイン，ストリーム型ソケット）による通信では，ファイルシステムにソケットファイルを作成し，そのファイルを介して通信を行う．ソケットファイルは，サーバとクライアントで共通のファイルを使用する必要がある．

```
int main(int argc, char* argv[])
{
  char   strLine[MAXLENGTH];
  int    iFdSock, iFdCri;
  int    iLength;
  int    iReturn;
  struct sockaddr_un sockAddress; // ソケット用
  struct sockaddr_un criAddress; // クライアント用
```

- **struct sockaddr_un sockAddress**
- **struct sockaddr_un criAddress**

  sockaddr_un 構造体は, UNIX LOCAL なソケットのアドレスの情報を表す構造体であり, 以下のメンバで構成される.

```
struct sockaddr_un {
    u_char sun_len;          /* 構造体のサイズ */
    u_char sun_family;       /* ソケットファミリーネーム */
                             /* Step4-2 では PF_UNIX を指定 */
    char   sun_path[104];    /* ソケットファイルのパスを指定 */
};
```

なおサーバプログラムでは, ソケット用とクライアント用の二つの sockaddr_un 構造体を用意する必要がある.

```
  iReturn = MAXLENGTH;

  // socket の作成. UNIX ドメイン, ストリーム型ソケット
  if( (iFdSock = socket(PF_UNIX, SOCK_STREAM, 0)) == -1){
    printf("Socket error.¥n");
    exit(-1);
  }
```

1. socket()によるソケット生成

- **iFdSock = socket(PF_UNIX, SOCK_STREAM, 0);**

  PF_UNIX, SOCKSTREM 型のソケットを生成する. 戻り値 iFdSock はソケットの識別子である. 以降の通信にはこのソケット識別子を用いる. 第一引数 PF_UNIX は, ローカル通信を行うことために指定するソケットファミリーネームを表す. 第二引数 SOCK_STREAM は, 順序・信頼性付きの双方向バイトストリームを使用することを表す. 第三引数 0 は, ソケット固有のプロトコルを使用することを表す.

```
// sockaddr_un 構造体を 0 で初期化
bzero((char*)&sockAddress, sizeof(sockAddress));
```

2. bzero()による sockaddr_un 構造体の初期化

- **bzero((char*)&sockAddress, sizeof(sockAddress));**
  ソケットプログラムでは bzero()関数を用いて sckaddr_un 構造体を 0 で初期化するのがお約束である.

```
// socket の名前を設定
sockAddress.sun_family = AF_UNIX;
strcpy(sockAddress.sun_path, SOCKETNAME);
```

3. sockaddr_un 構造体の設定

- **sockAddress.sun_family = AF_UNIX;**
  ソケットを用いたローカル通信では, sockaddr_un 構造体のメンバ sun_family に AF_UNIX を指定するのがお約束である.

- **strcpy(sockAddress.sun_path, SOCKETNAME)**
  sockaddr_un 構造体のメンバ sun_path にソケットファイルのパスを設定する.

```
// ソケットファイルをアンリンク
// blind()のエラー防止用
unlink(SOCKETNAME);
```

4. unlink()によるソケットのアンリンク

- **unlink(SOCKETNAME);**
  ソケットを用いたローカル通信では, bind()でソケットファイルを登録する前に unlink()を行うのがお約束である.

```
// アドレスをソケットにバインド
if(bind(iFdSock,
        (struct sockaddr*)&sockAddress,
         sizeof(sockAddress.sun_family) + strlen(SOCKETNAME)) == -1){
  printf("Bind error.¥n");
  exit(-1);
}
```

5. bind()によるソケット登録

・**bind(iFdSock,**
(struct sockaddr*)&sockAddress,
sizeof(sockAddress.sun_family) + strlen(SOCKETNAME))

ソケット識別子 iFdScok を持つソケットを，アドレス sockAddress を持つソケットとして登録する．これは伝統的に「ソケットに名前をつける」と呼ばれる．戻り値が 0 なら成功を，-1 なら失敗を表す．第一引数 iFdScok は接続を行いたいソケットのソケット識別子を表す．第二引数 scokAddress はソケットのアドレスなどを表す構造体を表す．第三引数 sizeof(sockAddress.sun_family) + strlne(SOCKETNAME)は sockAddress 構造体の大きさを表す．

```
// ソケットをリッスン
if(listen(iFdSock, 1) == -1){
  printf("Listen error.¥n");
  exit(-1);
}
```

6. listen()によるソケットの接続準備

・ **listen(iFdSock, 1)**

ソケットをクライアントからの接続準備の状態にする．戻り値が 0 なら成功を，-1 なら失敗を表す．第一引数 iFdScok は接続を行いたいソケットのソケット識別子を表す．第二引数 1 は，サーバに接続可能な待ち行列の最大数を表す．

```
// 無限ループ
for( ; EVER; ){
  iLength = sizeof(criAddress);
```

```
   if((iFdCri = accept(iFdSock, (struct sockaddr*)&criAddress, &iLength)) ==
-1){
     printf("Accept error.¥n");
     exit(-1);
   }
```

7. accept()によるクライアントからの接続待機

・accept(iFdSock, (struct sockaddr*)&criAddress, &iLength)
ソケットからの接続要求を受け入れるための待機状態に入る. 戻り値はクライアント識別子を表す. 第一引数 iFdScok は接続を行いたいソケットのソケット識別子を表す. 第二引数 scokAddress はソケットのアドレスなどを表す構造体を表す. 第三引数&iLengthh は sockAddress 構造体の大きさを表す.

```
   // クライアントと接続されているソケットからデータを受け取る
   while(iReturn = read(iFdCri, strLine, MAXLENGTH)){
     printf("%s", strLine);
   }
```

8. read() / write()によるクライアントとのデータ送受信.

・read(iFdCri, strLine, MAXLENGTH);
read()関数によりクライアントからメッセージを受信する. 第一引数 iFdSock はソケット識別子を表す. 第二引数 strLine は受信用バッファを表す. 第三引数 MAXLENGTH は受信用バッファ strLine のサイズを表す. なおステップ 4-1 のプロセス 2 はメッセージの送信は行わないので, write()は使用しない.

```
   close(iFdSock);
   close(iFdCri);
```

6. close()によるソケットのクローズ.

・ close(iFdSock);
・ close(iFdCri);
socket()によって作成したソケットは必ず close()で閉じる必要がある. これは, fopen()で開いたファイルを fclose()で閉じるのと同じ概念である.

```
  }
   return 0;
}
```