Makefile, TCP ソケットサーバ, コマンドライン引数

樋口さぶろお

龍谷大学理工学部数理情報学科

応用プログラミング☆実習 L11(2017-12-12 Tue)

最終更新: Time-stamp: "2017-12-22 Fri 12:28 JST hig"

今日の目標

- TCP ソケットサーバの (基本的な) 処理の流れを説明できる
- 分割コンパイルを make で自動化できる
- コマンドライン引数を読む TCP ソケットクライアントが書ける

http://hig3.net

先週の課題から I

ポインタ

配列変数とは, 先頭要素を指すポインタ変数 **void** die(**char** *message) と **void** die(**char** message[]) は同じ.

引数あり関数の役立つところ

似た処理の記述を一か所にまとめる. 異なる部分は引数に押し込める.

main.c

```
1 #include <math.h>
2       /*中略*/
3       double y,z;
4       y=1.3*exp(1.3);
5       printf("%f\n",y);
6       z=-3.2*exp(-3.2);
7       printf("%f\n",z);
```

lib.h

```c
#ifndef LIB_H
#define LIB_H
double f(double x);
#endif
```

main.c

```c
#include "lib.h"
/*中略*/
    double y,z;
    y=f(1.3);
    z=f(-3.2);
```

lib.c

```c
#include <math.h>
#include "lib.h"

double f(double x){
    double a;
    a=x*exp(x);
    printf("%f\n",a);
    return a;
}
```

ここまで来たよ

10 エラー処理・分割コンパイル・コマンドライン引数

11 Makefile, TCP ソケットサーバ, コマンドライン引数
- TCP ソケットサーバ
- make と Makefile

TCP ソケットクライアントの処理の流れ (再)

ソケット通信	(ファイル入力での例え)	関数
ソケット作成	fopen	socket
接続	fopen	connect
読み取り/書込	fscanf,fprintf	recv,send
接続終了	fclose	shutdown
ソケット廃棄	fclose	close

fopen-fscanf-fclose でなく, 低レベルファイル入出力 open-read-close のほうがよく対応する.

TCP ソケットサーバの処理の流れ

	ソケット通信	クライアント	サーバ
s1	ソケット作成		s=socket
s2	サーバポート指定		bind
s3	待ち受け		listen
c1	ソケット作成	sc=socket	
c2	サーバに接続	connect	
s4	接続受入, 通信ソケット作成		ws=accept
c3s	通信 (両方向)	send/recv	recv/send
c4s	接続終了	shutdown(sc)	shutdown(ws)
c5s	ソケット廃棄	close(sc)	close(ws)
s5	接続受入再開		s4 にもどる

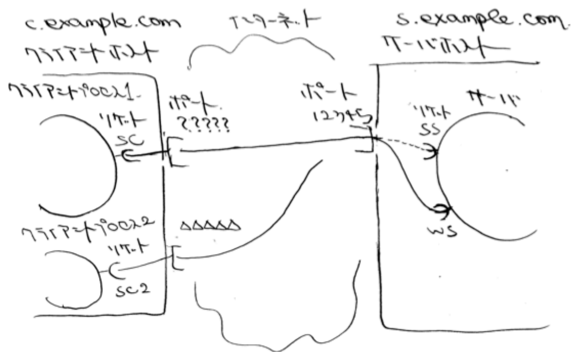
課題 socket-server01

待受ソケット `s` とは別の通信用ソケット `ws` を作る. `s` は永久に待受を続ける.

ブロック, sleeping OS でプロセスが実行待ちの状態. 入力がくる, 出力を受け入れてもらえるのを待つ.

計算機システム II

ひとつのクライアントと通信している間に別のクライアントが通信してくると, 待ち行列に入れられる. その間, クライアントのプロセスはブロックする. 計算機システム II 多くのクライアントを同時に相手にするには, サーバが fork システムコールで分身すればいい.


c.example.com
インターネット
s.example.com
クライアントホスト
サーバホスト
クライアントプロセス1
ポート
?????
ポート
12345
ソケット
sc
ソケット
ss
サーバ
クライアントプロセス2
△△△△△
ソケット
sc2
ws

現在のポートの状態を表示する netstat コマンド

```
1  $ netstat -at
2  稼働中のインターネット接続 (サーバと確立)
3  Proto 受信-Q 送信-Q 内部アドレス              外部アドレス                 状態
4  tcp        0      0 localhost.localdom:6010 *:*                          LISTEN
5  tcp        0      0 *:microsoft-ds          *:*                          LISTEN
6  tcp        0      0 s01cd0542-160:ssh       133.83.83.76:65008           ESTABLISHED
7  tcp        0      0 s01cd0542-160:ssh       ueh-36-42.st.ryuk:50423      ESTABLISHED
8  tcp        0      0 s01cd0542-160:59322     s01svvhd02.dream.r:http      ESTABLISHED
9  tcp        0      0 s01cd0542-160:36902     ifs3:nfs                     TIME_WAIT
10 tcp        0      0 s01cd0542-160:46408     ifs3:sunrpc                  TIME_WAIT
11 tcp        0      0 s01cd0542-160:59320     s01svvhd02.dream.r:http      TIME_WAIT
12 tcp        0      0 s01cd0542-160:773       ifs3:nfs                     ESTABLISHED
13 tcp        1      0 localhost.localdo:51020 localhost.localdoma:ipp      CLOSE_WAIT
14 tcp        0      0 s01cd0542-160:59321     s01svvhd02.dream.r:http      TIME_WAIT
15 ....
```

IP アドレスはコンピュータ (ホスト) を一意的に区別する, と言ったが例外あり. 127.0.0.1 localhost は '自分'. 'プライベートアドレス' 10.*.*.*, 192.168.*.*, (説明してない記法)172.16.0.0/12 は内線のようなもので, '閉じた' 組織内では自由に使ってよい.

socket-server01.c

```
1  #include <arpa/inet.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <sys/types.h>
6  #include <sys/socket.h>
7  #include <sys/uio.h>
8  #include <unistd.h>
9
10 #define NQUEUESIZE 5
11 #define SERVER_PORT 0 /* コンパイルはできるけど実行する際には
12 #define BUFFERSIZE 1000
13
14 /* 関数プロトタイプ宣言 */
15 void die(char message[]);
16
17 int main(void){
18   int s;/* 待受用ソケットのデスクリプタ*/
```

```
19      int ws;/* 通信用ソケットのデスクリプタ*/
20      struct sockaddr_in sa, ca; /* サーバ, クライアントアドレス
21      socklen_t ca_len; /* typedef で書かれた型 */
22      int messagesize;
23      char recvbuffer[BUFFERSIZE];
24      int val=1;
25      int n;
26      char message[BUFFERSIZE];
27
28      if( (s=socket(AF_INET, SOCK_STREAM,0))==-1){
29        die("socket");
30      }
31
32      /*呪文(魔術)*/
33      if(setsockopt(s,SOL_SOCKET,SO_REUSEADDR,&val,sizeof val)){
34        die("socketopts");
35      }
36
37      sa.sin_family=AF_INET;
38      sa.sin_port = htons(SERVER_PORT);
39      sa.sin_addr.s_addr = htonl(INADDR_ANY);/*IPアドレス指定無*
```

```
40     /*  sa.sin_zero[8];*/
41     if( bind(s, (struct sockaddr *)&sa, sizeof(sa))==-1){
42       die("bind");
43     }
44
45     if( listen(s,NQUEUESIZE)==-1 ){
46       die("listen");
47     }
48
49     while(1){
50       ca_len=sizeof(ca);
51
52       if((ws=accept(s,(struct sockaddr *)&ca, &ca_len))==-1){
53         die("accept");
54       }
55
56       if((messagesize = recv(ws,recvbuffer,BUFFERSIZE,0))<0){
57         die("recv");
58       }
59       recvbuffer[messagesize]='\0';
60
```

```
61      if(strncmp(recvbuffer,"count ",strlen("count "))!=0){
62        strcpy(message,"Penguin!\n");
63      } else {
64        n=strlen(recvbuffer)-strlen("count ");
65        if (n==0){
66          strcpy(message,"No Adelie Penguins!\n");
67        } else if (n==1){
68          strcpy(message,"One Adelie Penguin!\n");
69        } else {
70          sprintf(message,"%d Adelie Penguins!\n",n);
71        }
72      }
73      fprintf(stderr,"%s\n", recvbuffer);
74
75      if( write(ws, message, strlen(message))==-1){
76        die("write");
77      }
78
79      if( shutdown(ws, SHUT_RDWR)==-1 ){
80        die("shutdown");
81      }
```

```
82
83        if( close(ws)==-1 ){
84          die("close");
85        }
86
87      }
88
89      return 0;
90    }
91
92
93    /* 関数定義 */
94    void die(char message[]){
95      perror(message);
96      exit(1);
97    }
```

サーバ側関数解説 I

```
#include <sys/types.h>
#include <sys/socket.h>

/*ソケットを作るサーバクライアント共通*/
int socket(略);

/*ソケットにポート番号をつける*/
int bind(int s, struct sockaddr *sa, int sa_len);
/* 入力 s ソケットデスクリプタ, sa サーバソケットのソケットア
   返り値 成功0 失敗 -1 */

/*
int listen(int s,, int len_queue);
/* 入力 s ソケットデスクリプタ, len_queue 接続待ち行列の長さ
   返り値 成功0 失敗 -1 */


```

サーバ側関数解説 II

```
18 int accept(int s, struct sockaddr *ca, int *ca_len);
19 /* 入力 s ソケットデスクリプタ
20    出力 *addr にはクライアントのソケットアドレス, ca_len その
21    返り値は通信用ソケットデスクリプタ */
```

API

sizeof 演算子

int x;
のとき, sizeof (int), sizeof x はバイト長 (整数) を返す
常に sizeof(char)==1

サーバクライアント両側関数解説 (再)

```
#include <sys/socket.h>

int socket(int family, int type, int protocol);
/* ソケットを作り, デスクリプタを返す */

int shutdown(int sockfd, int how);
/* 後始末 */
```

```
#include <unistd.h>
int close(int socketfd);
/* 閉じる */
```

クライアント側関数解説 (再)

```
1  #include <sys/types.h>
2  #include <sys/socket.h>
3
4  int connect(int sockfd, struct sockaddr *address, int addrle
5  /* デスクリプタsockfd で指定されるソケットから,
6  ソケットアドレス構造体 address で指定される通信先に接続要求を
7  接続失敗なら負の返り値*/
8
9  int recv(int sockfd, char *buf, int len, int flags);
10 /* bufは受信バッファ, len は受信最大バイト数 */
11 /* 返り値は受け取ったバイト数.　終了なら0, 失敗なら−1 */
12
13 int send(int sockfd, char *buf, int len, int flags);
14 /* bufは送信データ, len は送信バイト数 */
15 /* 返り値は送信したバイト数. 失敗なら−1 */
```

文字列を整数として解釈する方法 (復習)

課題 socket-client03

```
1     char s[]="10 penguins";
2     int n;
3     sscanf(s,"%d",&n);
```

```
1     char s[]="10 penguins";
2     int n;
3     n=atoi(s); /* Ascii character TO Integer */
4                /* atol, atod などもある */
```

エラーが起きないとすればどちらもあまり変わらない (が, s が外部/ユーザから来る場合は, エラーが起きないと仮定できないのだった).

ここまで来たよ

10 エラー処理・分割コンパイル・コマンドライン引数

11 Makefile, TCP ソケットサーバ, コマンドライン引数
- TCP ソケットサーバ
- make と Makefile

分割コンパイルはややこしい — make

課題 make01

```
1 $ cc -o die.o -c die.c  # die.c または die.h が更新されたとき
2 $ cc -o main.o -c main.c # main.c または die.h が更新されたと
3 $ cc -o main die.o main.o -lm # main.o または die.o が更新さ
```

依存性と手順をファイルにメモ. 後はそんなの計算機にまかせちゃえ

```
1 $ make main
```

ただし, 同じディレクトリ内に依存性と手順をメモしたテキストファイル Makefile を準備. Makefile はエディタで編集する.

Makefile

Makefile

```
1  # TABによる字下げは必須.
2  die.o: die.h die.c
3      cc -o die.o -c die.c
4
5  main.o: die.h main.c
6      cc -o main.o -c die.c
7  main: die.o main.o
8      cc -o main die.o main.o -lm
9
10 hello.o:hello.c
11      cc -o hello.o -c hello.c
12 hello: hello.c
13      cc -o hello hello.o
```

Makefile の一般的記法

ルール群を書く. 行の順序は自由. ターゲット間の空行自由.

```
1 ターゲット1: 依存ファイル11, 依存ファイル12 依存ファイル13
2 (TAB)ターゲット1を作るための手順1a
3 (TAB)ターゲット1を作るための手順1b
4
5 ターゲット2:....
```

```
1 $ make ターゲット1
```

とされたとき, まず 再帰的に

```
1 $ make 依存ファイル11 依存ファイル12 依存ファイル13
```

相当のことを行う. その後, 依存ファイル 11, 依存ファイル 12, 依存ファイル 13 のいずれかが, ターゲット 1 より新しいときだけ, 手順 1a, 1b を実行する.

*.c と *.h を作る手順, などは make がデフォルトルールを持っているので, 省略できることがある. 課題では無理に省略しなくてよい.

```
1 # 依存性だけは人間が教える必要
2 die.o: die.h die.c
3 main.o: die.h main.c
4 hello.o:hello.c
5 hello: hello.c
6
7 # -lm が必要かどうかは人間が指定する必要
8 main: die.o main.o
9     cc -o main die.o main.o -lm
```

Makefile を書き替えても, 依存性から求められなければ再コンパイルは起きない

強制的にコンパイルしたいとき (例:警告を見たい), 依存ファイルを強制的に新しくする.

```
1 $ touch main.c # main.c の最終更新時刻を現在とする
```

お知らせ

- 樋口オフィスアワー月 $3.5 - 4.5$(1-502), 金 4(1-502)
- ごめんなさい 2017-11-28 火 12 休講分を 2017-12-26 火 [ ] 講時に補講.
- L12 の最初でも非参照のテストやります… 出題計画は授業中に修正するかも.
  - ~~Makefile についての問題~~
  - TCP ソケットサーバについての多肢選択 or 正誤問題
  - コマンドライン引数のプログラミング的な問題
- Learn Math Moodle に予習問題を載せるので, それで準備してね.