エラー処理・分割コンパイル・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 L10(2017-12-05 Tue)

最終更新: Time-stamp: "2017-12-17 Sun 11:59 JST hig"

今日の目標

- プロトコルとは何かを説明できる
- エラー処理が読み書きできる.
- ファイル分割でプログラムを整理できる
- recv/send 両方を含むクライアントを書ける
- コマンドライン引数を読むプログラムを書ける

http://hig3.net

先週の復習 I

- 通信相手の指定方法 swallow.math.ryukoku.ac.jp:13 = 133.83.83.6:13 = ホスト名:ポート番号
  - ホスト名や IP アドレスはインターネット上のコンピュータを指定
  - ポート番号はコンピュータ上のプロセス (の待ち受け口=ソケット) を指定
    - ★ 有名なサービスは特定のポート番号 (well-known ports) を使っている
- いろんなサーバが, 各自のホスト名:ポート番号で待ち受け中. クライアントはホスト名:ポート番号を指定してサービスをリクエストし, レスポンスを得る.
- ソケットによるクライアントの通信手順 socket, connect, recv/send, shutdown, close.

ここまで来たよ

9 IP アドレスとポート・クライアントサーバモデル

10 エラー処理・分割コンパイル・コマンドライン引数
- プロトコル とソケットクライアント
- エラー処理
- 関数定義と分割コンパイルと
- コマンドライン引数の扱い

プロトコル

先週の daytime サービス

ルール: クライアントがサーバに接続すると, サーバが, ”2017-12-03 Sun 13:27” のような文字列をクライアントに送る. ⇝ RFC

Protocol (プロトコル)

手順書. 手順やデータ形式の規約.

> TCP, IP, HTTP などの最後の P は Protocol の P. HTTP プロトコルってのは重複した言い方

今週の penguin サービスのプロトコル

クライアントが `count ` で始まる文字列を送ると, サーバはそれに続く文字数 (c) を数え, `c Adelie Penguins!` をクライアントに送る. それ以外の文字列なら `Penguins!`を送る.

> 実験, テスト用として有名なプロトコル: echo

> インターネットのプロトコルの王者: http(s). その上に, プロトコル, Web API がある.

エラー処理つきソケットクライアント (送受信)

```
1  #include <stdio.h>
2  #include <string.h> /* memset */
3  #include <sys/socket.h> /*socket, connect */
4  #include <arpa/inet.h> /* htons */
5  #include <netinet/in.h> /* htons, inet_addr */
6  #include <unistd.h> /* close sizeof */
7  #include <stdlib.h>
8
9  #define SERVER_ADDR "127.0.0.1"
10 #define SERVER_PORT 22222
11 #define BUFFERSIZE 1024
12
13 int main(){
14
15    int s;
16    struct sockaddr_in sa;
17    int count;
18    char recvbuf[BUFFERSIZE];
```

```
19    char sendbuf[]="count something.";
20    int status;
21
22    /* ソケットを作る */
23    s = socket(AF_INET,SOCK_STREAM,0);
24    if( s==-1 ){
25       perror("socket");
26       exit(1);
27    }
28
29    /* 接続先のホスト:ポート を構造体に詰める */
30    memset(&sa, 0, sizeof(sa));
31    sa.sin_family=AF_INET;
32    sa.sin_port=htons(SERVER_PORT);
33    sa.sin_addr.s_addr=inet_addr(SERVER_ADDR);
34
35    /* 接続する = open */
36    status=connect(s, (struct sockaddr *)&sa, sizeof(sa));
37    if(status==-1){
38       perror("penguin connect");
39       exit(1);
```

```
40      }
41
42      /* 送る printf */
43      count=send(s, sendbuf, strlen(sendbuf), 0);
44      if( count==-1 ){
45        perror("penguin send");
46        exit(1);
47      }
48
49      /* 受け取る scanf */
50      count=recv(s, recvbuf, BUFFERSIZE-1, 0);
51      if( count==-1 ){
52        perror("penguin recv");
53        exit(1);
54      }
55
56      /* 終端文字を加えて文字列にして出力 */
57      recvbuf[count]='\0';
58      printf("%s",recvbuf);
59
60      /* close */
```

```
61
62      status=shutdown(s, SHUT_RDWR);
63      if( status==-1){
64          perror("penguin shutdown");
65          exit(1);
66      }
67
68      status=close(s);
69      if( status==-1){
70          perror("penguin close");
71      }
72
73      return 0;
74  }
```

ここまで来たよ

9 IP アドレスとポート・クライアントサーバモデル

10 エラー処理・分割コンパイル・コマンドライン引数
- プロトコル とソケットクライアント
- エラー処理
- 関数定義と分割コンパイルと
- コマンドライン引数の扱い

エラーの種類

- コンパイルエラー ex. syntax error 文法エラー
  - プログラムを書いた人だけの責任
- ランタイムエラー 実行時エラー ex. segmentation fault
  - プログラムの外部に原因
    - ★ プログラマが想定していない入力を, ユーザが与えた
    - ★ システムが作れるソケットの個数を超えた
    - ★ ソケットサーバの電源がはいっていなかった
    - ★ 実行中にだれかがネットワークケーブルを引き抜いた

なるべく悪影響を防ぎ続行する. 終了するならプログラマにデバッグに有用な情報を残して終了する

エラー処理の極端なケース: 異常終了

- 関数内での return 文は, 関数を終了して, 呼び出し側に返り値を返す. 負の数, 特に -1 が関数内のエラーを意味することがある. 0 は正常終了.
- stdlib.h で定義された関数 `void exit(int status)` は, プログラム全体を終了して, OS に返り値を返す. 0 は正常終了. それ以外は異常終了.

ソケットクライアントのエラー処理

特に socket, connect, recv, send に対してエラー処理が必要.

これらはたまたま, OS に対して処理を依頼するシステムコール計算機システム
II
(特定の OS 例えば Linux の) 様々なシステムコールの使い方を学ぶ授業/教科書 は (Linux) システムプログラミング という名前がついている.

エラーが起きたか, どんなエラーか判断する方法

- 関数やシステムコールの返り値でエラーか, どんなエラーか判断
- システコムールの場合, perror(3) の出力でエラーの内容を知る
  - 上のは, man 3 perror で調べろ, という意味 (だけど, サービスのため次のページ)

perror

perror(”プログラム内の位置の情報の入ったプログラマのメッセージ”);
とすると, ‘内部で’ エラーが起きたときに, 「errno + プログラム内の位置の情報の入ったプログラマのメッセージ + : + システムのエラーメッセージ」を標準エラー出力 stderr に出力する.

perror(3) のヘッダファイルと型

```
1 #include <stdio.h>
2 void perror(char *s);
```

システムコールのエラーの内容をもっと詳細に知りたい, エラーの種類によって分岐 (if-then) したい, というときには, **#include** <errno.h> して, グローバル変数 int errno , 関数 char *sterror(int errno) を利用する. 実は perror はこれらを内部に隠して使っている.

標準エラー出力

エラーの出力は, データの出力と分けて, 標準エラー出力へ.

fprintf (stderr ,”something”)　リダイレクト (計算機システム II)

簡潔に書こう

```
s=socket(...);
if(s==-1){
   /*エラー処理;*/
}
/* sがどんな値なら
どんなエラーか,
は socket(2) 参照 */
```

```
/* socket の返り値を後で使わないとき(非現
if( socket(...)==-1 ){   /*エラー処
理;*/}
```

```
/* socket の返り値をsに保存するとき*/
if( (s=socket(...)) ==-1 ){   /*エ
ラー処理;*/}
```

(a=f())==2 の括弧は必要. つけないと, a=(f()==2) の意味になり, a には等式の真偽値 0 または 1 が代入される (代入文の返り値で if は分岐する)

C では下品, 最近の言語では上品とされている書き方として,

```
if( -1==do_something() )   die();
```

は次のようにも書ける. || は論理和 OR. 'Get up or you will be late' 英語.

```
( (-1==do_something() ) || ( die() ));
```

根本的な解決 exception(例外), try-catch in Java　　グラフィックス基礎 (3 年前期)

ここまで来たよ

9 IP アドレスとポート・クライアントサーバモデル

10 エラー処理・分割コンパイル・コマンドライン引数
- プロトコル とソケットクライアント
- エラー処理
- 関数定義と分割コンパイルと
- コマンドライン引数の扱い

関数定義と分割コンパイルと

だいぶ簡潔になったとは言え, 毎回,

```
1 if(  実行()==失敗の値  ){
2     perror("この場所を表す文字列");
3     exit(1);
4 }
```

と書くことになる.

関数定義+分割コンパイルで解決 (復習)

```
1 perror("この場所を表す文字列");
2 exit(1);
```

を, "この場所を表す文字列" を引数とする関数 **void** die(**char** message[]) と定義しよう. この関数の関数プロトタイプ宣言を die.h, 定義を die.c に書き, Makefile を使って分割コンパイルしよう.

> Refactoring (リファクタリング) プログラムが同じ動作をするままの状態を保って, 関数や変数の定義を改善して, ソースを自分や他人にわかりやすく, その後の改造に適した構造に再編すること

ここまで来たよ

9 IP アドレスとポート・クライアントサーバモデル

10 エラー処理・分割コンパイル・コマンドライン引数
- プロトコル とソケットクライアント
- エラー処理
- 関数定義と分割コンパイルと
- コマンドライン引数の扱い

汎用クライアント telnet

汎用クライアント `telnet(1)`, `nc(1)`

```
telnet  host  port
nc –C   host  port
```

標準入力をサーバにリクエストとして送り, サーバからのレスポンスを標準出力に出す.

- telnet は Control と] の同時押し, 'quit'+Enter で終了.
- nc は Control と C の同時押しで終了.

```
telnet www.a.math.ryukoku.ac.jp 80
GET /
```

と, Web ブラウザの URL バーに `http://www.a.math.ryukoku.ac.jp` と入力し, ページ上で右クリックして「ソースを比較」したときの表示を比較しよう. 80 は何のポート番号?

m

コマンドライン引数

```
telnet      www.math.ryukoku.ac.jp  80
#argv[0]    argv[1]                 argv[2]     argc=3
```

コマンド名に続く www.mathr.ryukoku.ac.jp, 80 をコマンドライン (第 1, 第 2) 引数という.
コマンドがどのようなコマンドライン引数を「取る」か, 何に使うかは, man に書かれている.
特に, -c, --help のように '-' から始まるコマンドライン引数をコマンドラインオプションといい, コマンドの振る舞いを変更するのに使われる.

```
tail penguin-client.c
tail -30 penguin-client.c
tail -r -30 penguin-client.c
```

C プログラムでのコマンドライン引数の受け取り方

main の仮引数

```
int main(int argc, char *argv[]){ ... }
/* int main(int argc, char **argv){ ... } も同じこと */
```

argc コマンドライン引数の個数. コマンド名を含む.
argv キャラクタ型ポインタの配列=文字列の配列

	0	1	2	3	4	5	6		
argv[0]	t	e	l	n	e	t	\0		
argv[1]	w	w	w	.	m	a	t	···	\0
argv[2]	8	0	\0						

コマンドライン引数のサンプルプログラム

```
1  #include <stdio.h>
2  #include <string.h>
3  #define BUFSIZE 1000
4
5  int main(int argc, char *argv[]){
6  /* int main(int argc, char **argv){  としても同じこと*/
7  /* argv は, キャラクタ型のポインタの配列=文字列の配列 */
8
9    char message[BUFSIZE];
10   char andstring[]=" and ";
11
12   strcpy(message,argv[1]);
13   strcat(message,andstring);
14   strcat(message,argv[2]);
15
16   /* for test */
17   /* printf("%c\n",argv[0][0]); */
18   /* printf("%c\n",argv[0][1]); */
19   /* printf("%s\n",argv[0]); */
20
21   printf("%s\n",message);
22
23   return 0;
24 }
```

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 たぶん 12 月中に補講.
- L11 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.
  - ソケット, サーバ, クライアント, プロトコルについての選択肢的問題
  - エラー処理の簡潔な書き方をしたソースの動作を説明する
  - 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.