郵便番号サーバ実装例, strtok, static 変数, Makefile

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 L12(2017-12-19 Tue)

最終更新: Time-stamp: "2017-12-19 Tue 13:17 JST hig"

今日の目標

- この授業のゴールまでの計画を説明できる
- 郵便番号検索プロトコルが説明できる
- strtok で token を分解するプログラムが書ける
- Makefile が書ける

http://hig3.net

ここまで来たよ

先週までのあらすじと次回以降予告

- L09 ネットワークとは? クライアントサーバシステムとは?

  IP アドレス:ポート　sizeof　ifconfig　host

- L10 TCP ソケットクライアントの作り方と動作

  コマンドライン引数　エラー処理　分割コンパイル　関数化

- L11 TCP ソケットサーバの作り方と動作

  make　atoi

- L12 郵便番号探索サーバの例, システムとプロトコル設計

  strtok

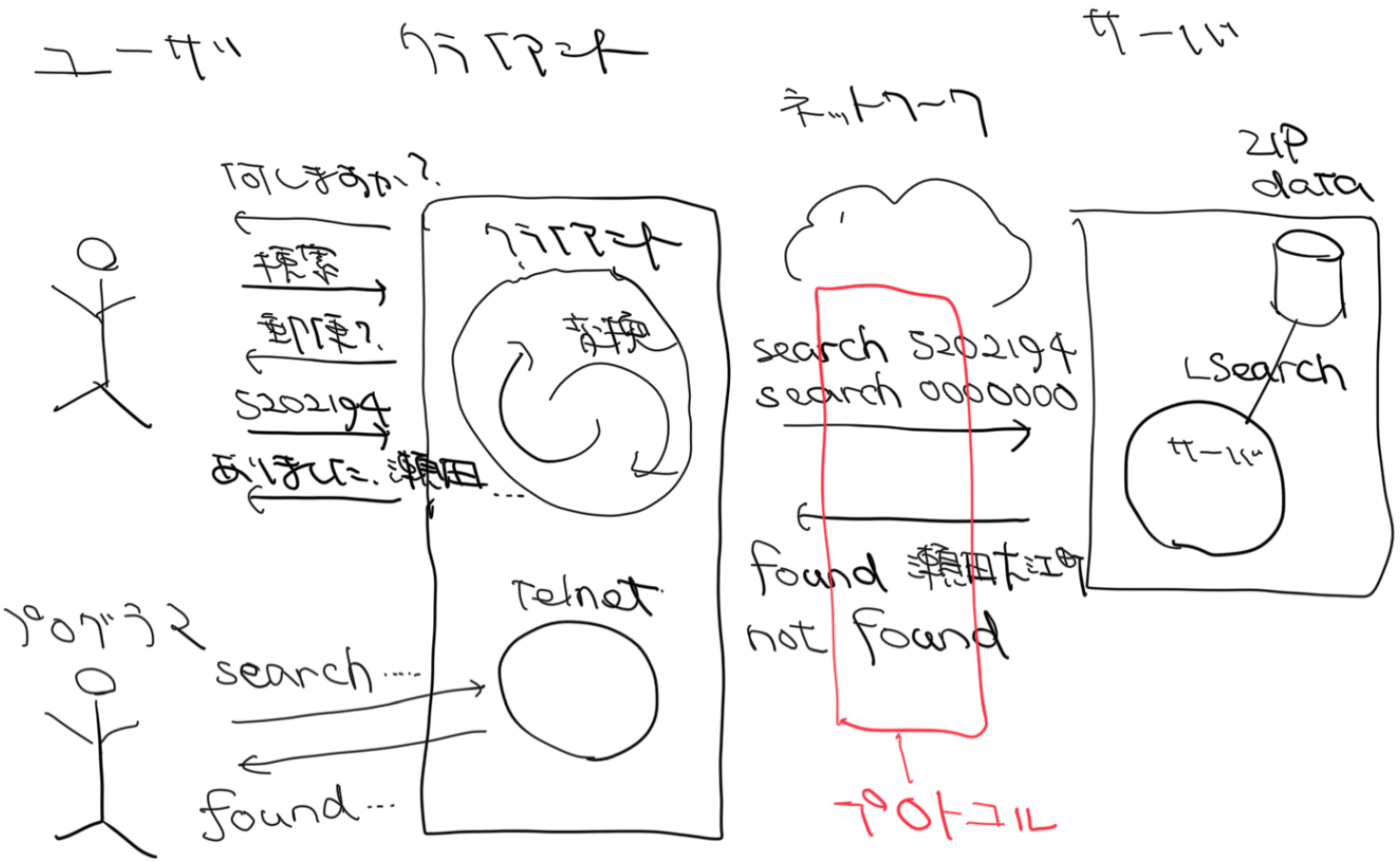
- L13 郵便番号探索クライアントの例, 1 回 accept, 複数回 send,recv するサーバの実装, システムとプロトコル設計

  fgets

- L14 システム制作 (1)
- L15 システム制作 (2)

以下のシステムの例をここ 2 回で説明する.
指定のチームで, このシステムの拡張または似たシステムを立案し (プロトコルを作り), (サーバ-クライアントを) 分担して作成する.

Zip fig



機能とプロトコル

機能	プロトコル		サーバ内の処理
	リクエスト	レスポンス	
1 個の郵便番号の住所を表示する	search 整数	found 郵便番号 住所 not found	int LSearch(...) を 1 回使用
—	上にあてはまらないもの	error	
⋮			

- クライアント-サーバ間の通信規則をあいまいさなく定めたもの．リクエストとレスポンスの意味とあいまいさのない書式
- プロトコルに従うなら, クライアント, サーバの実装を交換しても機能する．例: telnet 対 zip-client

ここまで来たよ

strtok の仕様と動作 I

サーバは, プロトコルに従った search 5202194　5200000 0000000 のようなリクエストを受け取る. 各単語に分解したい.

一般に, 文字列例えば "abc 10.3, def:32" を, スペース, コンマ, コロンを区切り文字 (デリミタ) として"abc","10.3","def","32" の 4 個のトークンに分解したい.

- token トークン それ以上分けたくない, 意味のあるかたまり
- delimiter デリミタ 区切り文字 トークンとトークンをわかる文字 (群)

strtok の仕様と動作 II

トークンもデリミタも, どんな順番で何が何個来るかわからない.

案 1 scanf("%s",s) 空白とや改行で止まるが, コロンは scanf("

案 2 scanf("%s␣%s␣%s␣%s",s1,s2,s3,s4) 変数の個数固定になっちゃう. 特定の区切り文字パターンにしか対応できない.

案 3 argv[n] コマンドライン引数にしか対応できない. 区切り文字は空白 (複数個可) 限定.

文字列の前から順に読んでいって, if-then の物量攻撃で行けば何とかなるはず. それをあらかじめ書いてくれたのが strtok 関数.

```
1 #include <string.h>
2 char *strtok(char given_string[], char delimiter[]);
```

strtok の仕様と動作 III

given_string 分割すべき文字列. NULL が指定されたら前回使ったものとみなされる.
delimiter 区切り文字を (順序を問わずに) 並べた文字列.
副作用 given_string の, 返り値の直後の delimiter が終端文字に書き替えられる.
返り値 今回返す token. いまの注目点が終端記号なら NULL.

この関数は, "a::b" から, "a","b" を取り出し, 2 番目の空文字列は取り出さない. ::が 1 個の区切りと見なす.

strtok の動作の様子

```
1 char t[]="abc 10.3,   def:32";
2 char d[]=" \t:,";
```

t																	
a	b	c	␣	1	0	.	3	,	␣	␣	␣	d	e	f	:	3	2	\0

s=strtok(t,d);

st																	
a	b	c	\0	1	0	.	3	,	␣	␣	␣	d	e	f	:	3	2	\0

s=strtok(NULL,d);

t				s													
a	b	c	\0	1	0	.	3	\0	␣	␣	␣	d	e	f	:	3	2	\0

s=strtok(NULL,d);

t												s						
a	b	c	\0	1	0	.	3	\0	␣	␣	␣	d	e	f	\0	3	2	\0

s=strtok(NULL,d);

t															s			
a	b	c	\0	1	0	.	3	\0	␣	␣	␣	d	e	f	\0	3	2	\0

s=strtok(NULL,d);

t																		
a	b	c	\0	1	0	.	3	\0	␣	␣	␣	d	e	f	\0	3	2	\0

, s=NULL

引数 NULL, d だけでこんなことできっこない. 仏様が s を内証で教えてる?

strtok を使ったサンプル I

```
1  #include <stdio.h>  /* for printf */
2  #include <string.h> /* for strtok */
3
4  void analyze(char *line);
5
6  int main(){
7    char test1[]="ab c";
8    char test2[]="ab: cd e";
9    char test3[]="ab: cd\ne:f ";
10
11   analyze(test1); /* 直に analyze("ab c") などとするとだめ*/
12   analyze(test2); /* なぜなら… */
13   analyze(test3);
14
15   return 0;
16 }
17
```

strtok を使ったサンプル II

```
18 void analyze(char *line){
19   char delimiters[]=" \t:"; /*デリミタ群 \t はタブ*/
20   char *s;
21
22   printf("[given]'%s'\n",line);
23
24   s=strtok(line,delimiters);
25   while( s!=NULL ){
26     printf("[decomposed]'%s'\n",s);
27     s=strtok(NULL,delimiters);
28   }
29   return;
30 }
```

実際に使うときは, main(や上位の関数) の中で strtok と, 得られた token の処理 (整数に変換して検索するとか) と混ぜて実行するでしょう.
analyze のような関数にはならないでしょう.

static 変数 (strtok の内部を想像) I

いま扱ってる文字列変数 (=キャラクタへのポインタ) をおぼえているはず
strtok の定義内では

```
1 static char *s
```

が使われている.

静的 (static)

関数内でローカル変数を宣言する際に static をつけると, 記憶クラス (寿命) が通常と異なり, 関数が終了しても値が保持され, 次回の関数呼び出しの際に値を引き続き利用できる.

```
1     int f(int x){
2       static int sum=0;
3       return sum=sum+x;
4     }
```

static 変数 (strtok の内部を想像) II

使用例

```
1    printf("%d\n",f(10)); /* これが最初の f(int x) 呼出 */
2    printf("%d\n",f(10));
3    printf("%d\n",f(10));
```

出力

キーワード static は, 別のところ (例えばグローバル変数や関数) で別の意味になる別の使い方がある

static 変数は, メモリ上の静的領域に置かれる　記号処理 (3 年前期)

ここまで来たよ

分割コンパイルはややこしい — make (再度)

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

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

```
$ 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.
Emacs なら, C ファイルを編集しているときに, メニュー >Tools>Compile でミニバッファ(ウィンドウ下) に出てくるコマンドを修正して enter するのが便利. エラーが出たらその行に飛べる.

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 23 講時に補講 (1-542). 3 講時が実習. 昼休みにお弁当食べられる休憩室を確保: 1-534.
- L13 の最初でも非参照のテストやります
  - static 変数の振る舞い
  - strtok の振る舞い
  - Makefile の書き方
- Learn Math Moodle に予習問題を載せるので, それで準備してね.