演習1の解答例の解説

2004年10月21日

海谷 治彦

fork2.c の概要 shellの枠組プログラム

```
 1| main(int argc, char* argv[]){
 2| pid_t ch; char buf[100];
 3|
 4|   while(fgets(buf, 100, stdin)!=NULL){
 5|     buf[strlen(buf)-1]='¥0';
 6|     if((ch=fork())==0){ // child
 7|       execl(buf, buf, NULL); // execveを呼ぶ
 8|     }else if(ch>0){ // parent
 9|       sleep(10);
10|       printf("done %d¥n", ch);
11|       wait(0);
12|     }
13|   }
14|
15| }
```

execvの仕様

- 関数名 execve
- 返り値 int 成功する場合，返らない．失敗すると–1が返る.
- 引数 3つ
  - const char *filename プログラムの完全パス名
  - char *const argv[] コマンド名も含めた引数のリスト.
  - char *const envp[] 環境変数名と値の対のリスト，mainの引数と同じ.
  - 尚，argv, envpはNULLで終わっている必要がある.
- 機能: 呼び出したプロセスを引数で指定したプログラムのプロセスに書き換える.

使用例

```
#include <stdio.h>
#include <unistd.h>

main(){
  char* filename="/bin/ls";
  char* argv[4]={
    "ls", "-l", "/", NULL
  };
  char* envp[3]={
     "PATH=/sbin:/usr/local/bin", "LANG=ja_JP", NULL
  };

  execve(filename, argv, envp);
}
```

演習1の私の解決方針

- execveの簡易版，execvpを使おう.
  - 引数がほとんど同じ.
  - 可変長引数に対応し易い.
- パスは外部変数environを利用しよう.
- 問題は単なる文字列をコマンドと引数の文字列配列にバラすことだ.

execvpの仕様

- 関数名 execvp
- 返り値 int 成功する場合，返らない．失敗すると–1が返る.
- 引数 2つ
  - const char *filename プログラムの完全パス名
  - char *const argv[] コマンド名も含めた引数のリスト.
  - 尚，argvはNULLで終わっている必要がある.
- 外部変数
  - char** environ 環境変数のリストが入る
- 機能: 呼び出したプロセスを引数で指定したプログラムのプロセスに書き換える.

サンプルプログラムと結果

```
#include <stdio.h>

main(int argc, char* argv[], char* envp[]){
char** ptr;
        for(ptr=envp; *ptr!=NULL; ptr++){
                printf("<%s>¥n", *ptr);
        }
}
```

```
<USER=kaiya>
<LOGNAME=kaiya>
<HOME=/home/kaiya>
<PATH=/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin>
<SHELL=/bin/tcsh>
<HOSTTYPE=i386-linux>
<VENDOR=intel>
<OSTYPE=linux>
<MACHTYPE=i386>
<TZ=Japan>
<LANG=ja_JP.eucJP>
```

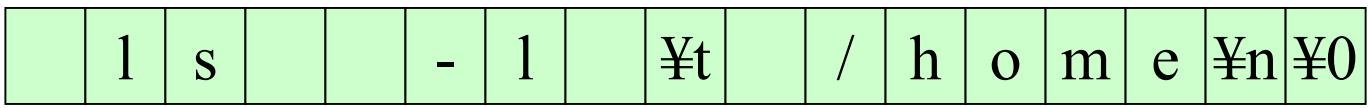
environの設定，実行方法の決定

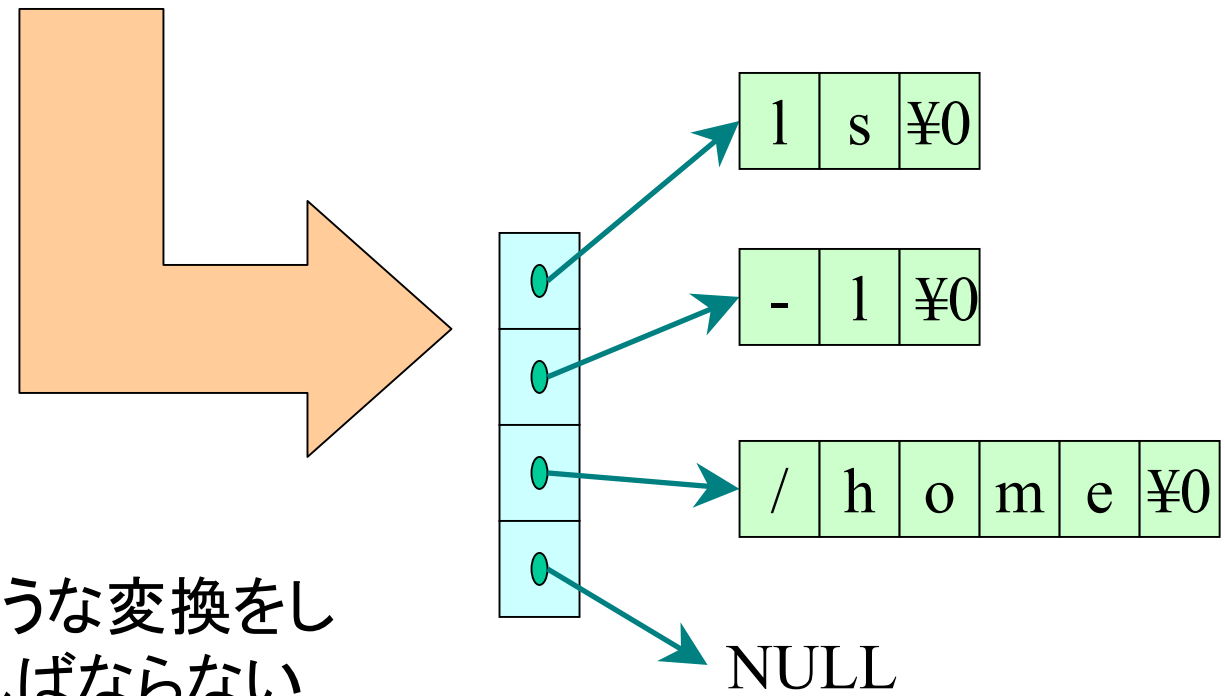
```
extern char** environ;

main(int argc, char* argv[]){
    environ=argv+1; // argv[0]はコマンド名自体
    ・・・・ // 以下略
}
```

ってな感じで，
./a.out PATH=/bin:/usr/bin LANG=dutch TZ=EST
とかいう形で実行．

文字列を文字列配列に

```
ls  -l  /home [エンター]
```

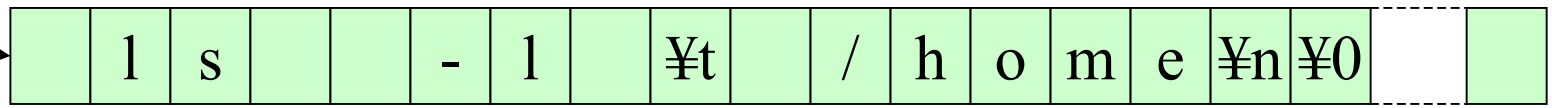




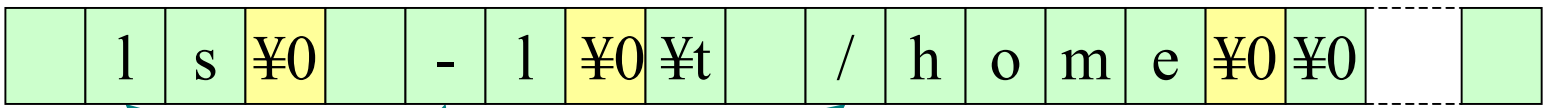
このような変換をし
なければならない.

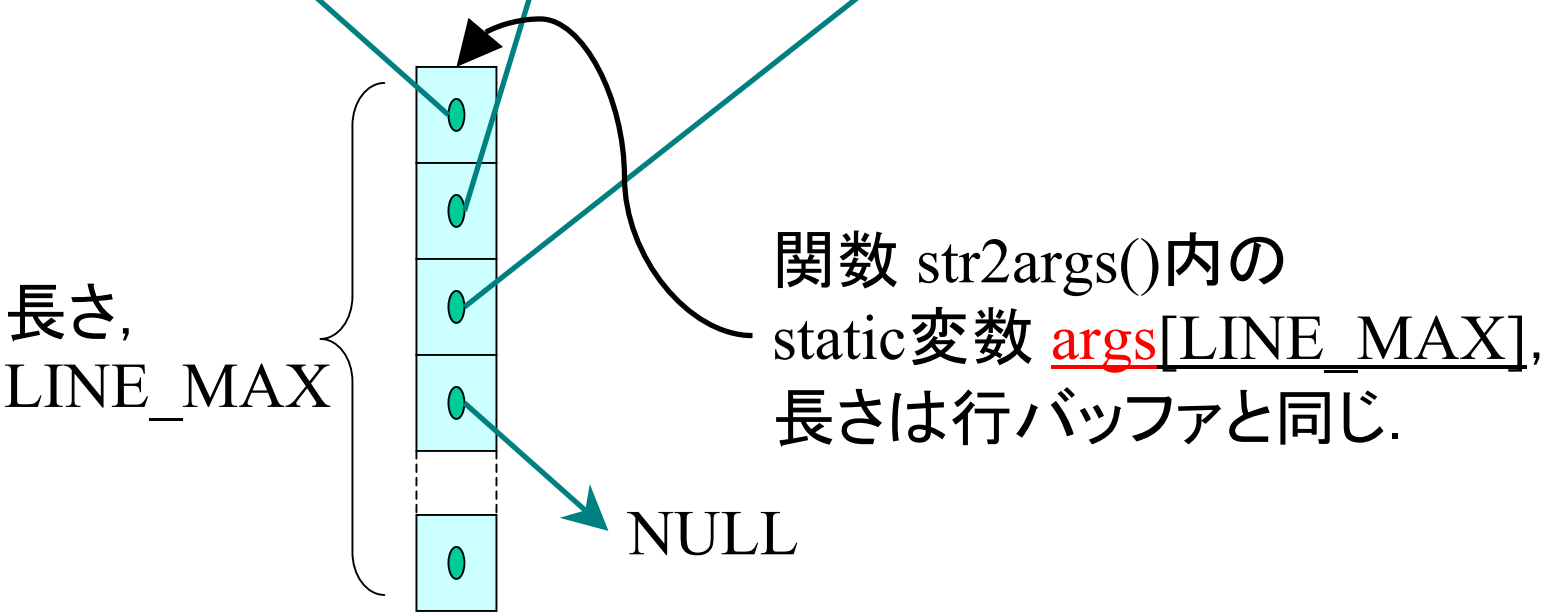
ex1a.c の実装

main関数内の行バッファ buf[LINE_MAX]



項目直後のセルに¥0を埋める.

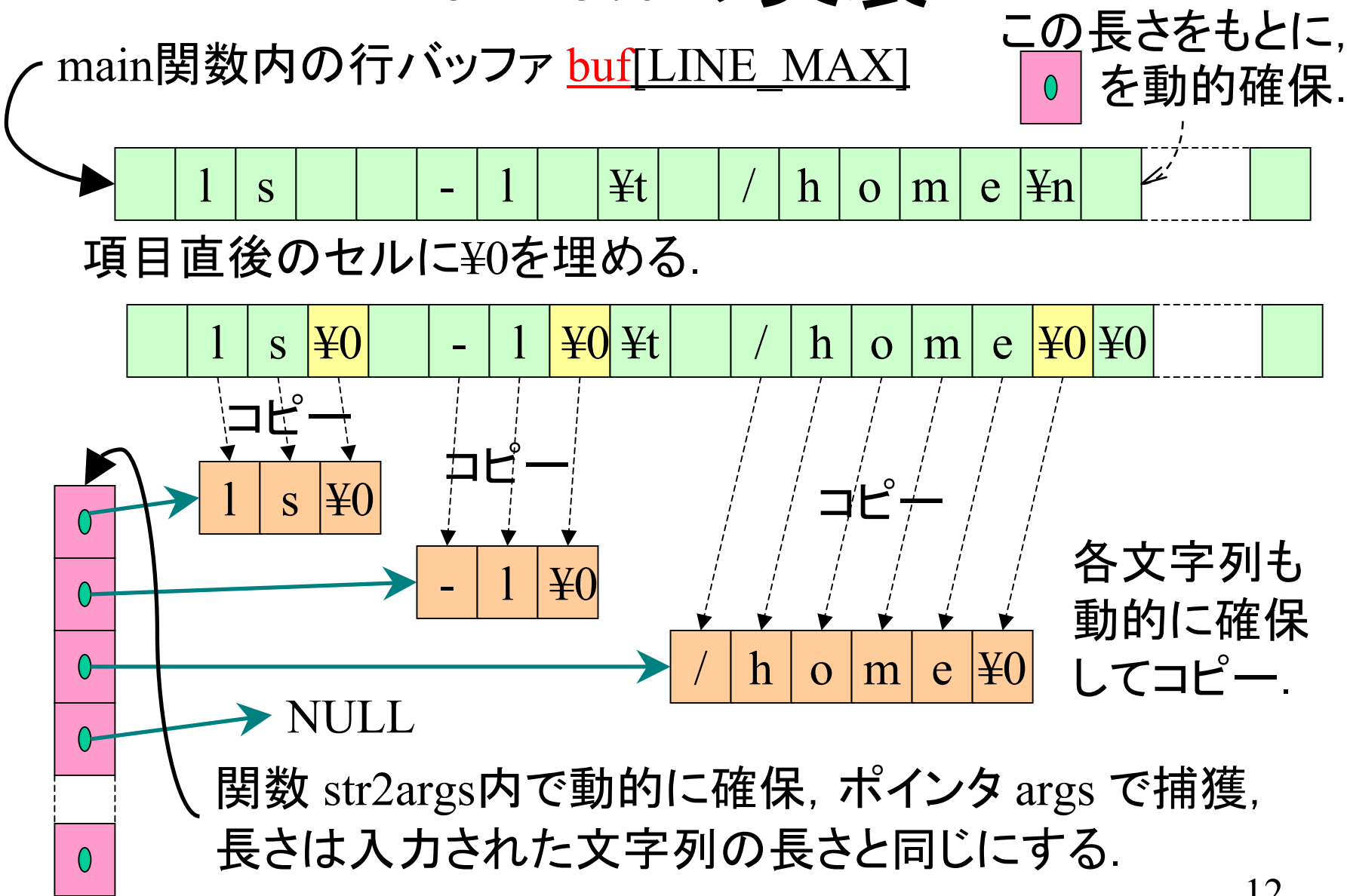




説明・考察

- mallocとか使わないでいいから楽.
- 関数内static変数を巧みに使っている.
  - が, しかし, コレが結構アブない.
  - 関数が並行して複数呼び出された場合危険.
- 行バッファの中身をそのまま流用.
  - コレも実はあぶない.
  - execvp呼び出しまでbuf[]内部を壊さないことを保障する必要がある.
- 文字列の配列のための行列 args[] は buf[]と同じ長さなので, 足りなくなることはありえない.
  - コマンドとオプションは空白文字で区切られるため, buf長より多いオプションが与えられることは論理的にありえない.
  - とはいえ, 無駄に長いのも確か.

ex1b.cの実装


この長さをもとに,
を動的確保.
main関数内の行バッファ buf[LINE_MAX]
l s - l ¥t / h o m e ¥n
項目直後のセルに¥0を埋める.
l s ¥0 - l ¥0 ¥t / h o m e ¥0 ¥0
コピー
コピー
コピー
l s ¥0
- l ¥0
/ h o m e ¥0
NULL
各文字列も
動的に確保
してコピー.
関数 str2args内で動的に確保, ポインタ args で捕獲,
長さは入力された文字列の長さと同じにする.

説明・考察

- ex1aよりは正統派.
- それでも入力時の内容 buf[]の一部を改変する.
- argsで参照する文字列配列もex1aほどは無駄ではない，とはいえ，結構無駄.
- malloc()を使うので結構長い.
- execvpを呼ぶともう制御が返ってこないか，もしくはexit()なのでfreeは実は必要ない.
  - しかも，fork()後にstr2buf()を呼ぶので，親側では気にしなくて良い.

パラっと皆さんの解答を見て

- 割と関数分けしてない人が多い.
  - 解答例にあるように関数わけしたほうが，この程度のプログラムでも見通しが良い.
- ま，リハビリになったでしょうか？
- fork(), exec()のからくりがなんとなくわかればOKかと思います.
- オンラインマニュアルやネットで色々調べましょうね.