@toc

简单的shell的实现

目的：主要就是为了加深对shell的底层原理的理解

```
[xifeng@VM-16-14-centos linux-learning]
[xifeng@VM-16-14-centos linux-learning]
[xifeng@VM-16-14-centos 简易版的shell]
total 0
[xifeng@VM-16-14-centos 简易版的shell]
```

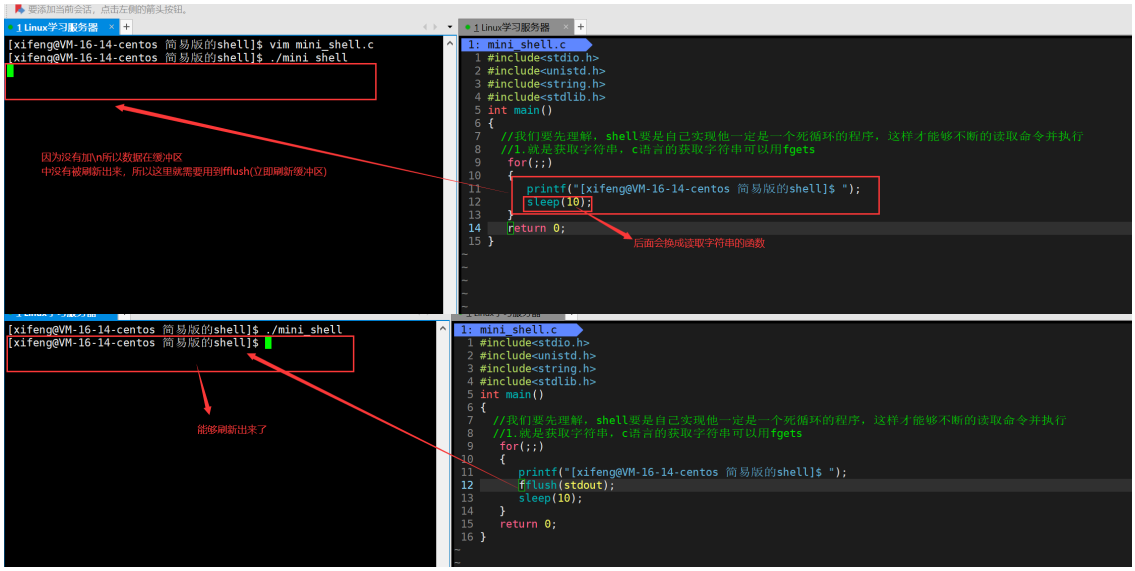
可以通过接口可以获取到这些主机名等等这里直接就用字符串打印了

- 当我们打开一个c文件默认就打开了三个输入输出流：**stdin(标准输入),stdout(标准输出),stderr(标准错误)**

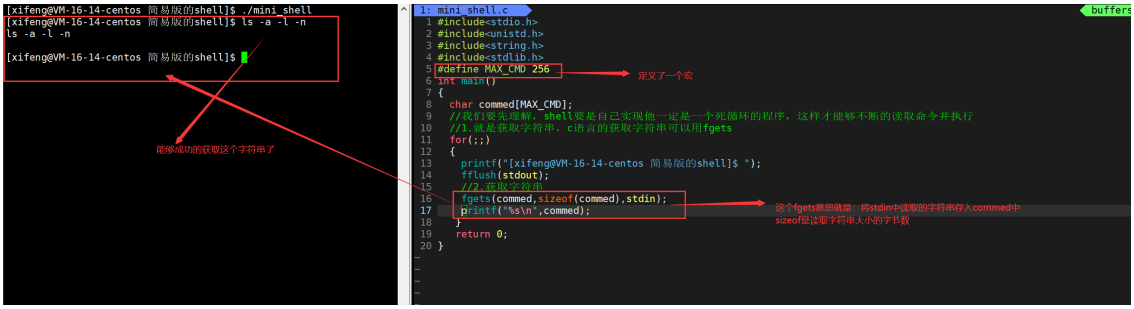
```
STDIN(3)            Linux Programmer's Manual           STDIN(3)

NAME
       stdin, stdout, stderr - standard I/O streams

SYNOPSIS
       #include <stdio.h>

       extern FILE *stdin;
       extern FILE *stdout;
       extern FILE *stderr;
```

---

1. 打印提示符



2. 获取命令字符串



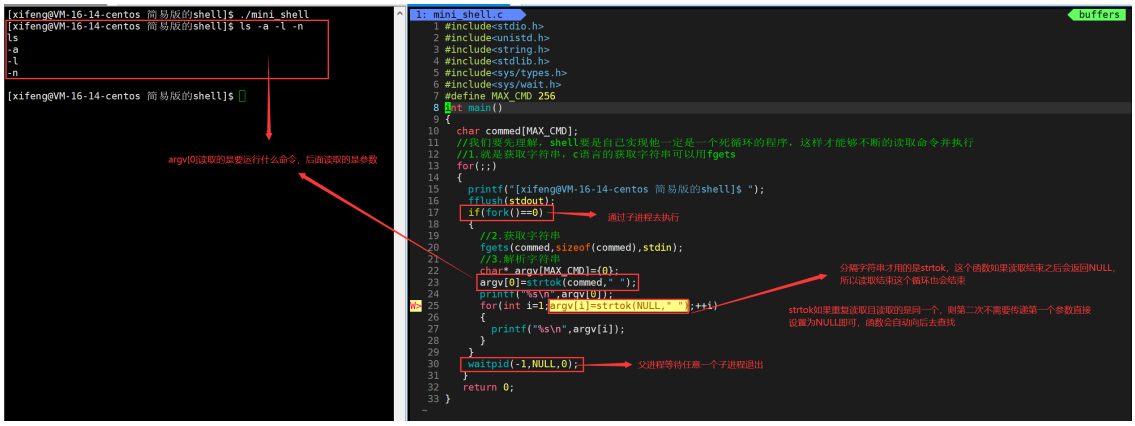
但是只是这样会有一个问题他会读取一个\n，也就是**出现了上面的ls -a -l -n下面空了一行，如何去掉这一行呢?**

```
commed[strlen(commed)-1]='\0';//即可
//strlen(commed)-1指向的是\n，这里是把\n替换成\0即可解决问题
```

- fgets()函数

```
[xifeng@VM-16-14-centos 简易版的shell]$ ^C
[xifeng@VM-16-14-centos 简易版的shell]$ clear
[xifeng@VM-16-14-centos 简易版的shell]$ man 3 fgets


GETS(3)            Linux Programmer's Manual           GETS(3)

NAME
       fgetc,  fgets,  getc,  getchar,  gets,  ungetc - input of
       characters and strings

SYNOPSIS
       #include <stdio.h>

       int fgetc(FILE *stream);

       char *fgets(char *s, int size, FILE *stream);

       int getc(FILE *stream);

       int getchar(void);

       char *gets(char *s);

       int ungetc(int c, FILE *stream);

DESCRIPTION
       fgetc() reads the next character from stream and  returns
       it  as  an unsigned char cast to an int, or EOF on end of
       file or error.

       getc() is equivalent to fgetc() except  that  it  may  be
       implemented  as  a macro which evaluates stream more than
       once.

       getchar() is equivalent to getc(stdin).

       gets() reads a line from stdin into the buffer pointed to
       by  s until either a terminating newline or EOF, which it
       replaces with a null byte ('\0').  No  check  for  buffer
       overrun is performed (see BUGS below).

       fgets()  reads  in  at most one less than size characters
       from stream and stores them into the buffer pointed to by
       s.   Reading  stops after an EOF or a newline.  If a new-
       line is read, it is stored into the buffer.  A  terminat-
       ing  null  byte ('\0') is stored after the last character
       in the buffer.

       ungetc() pushes c back to stream, cast to unsigned char
```

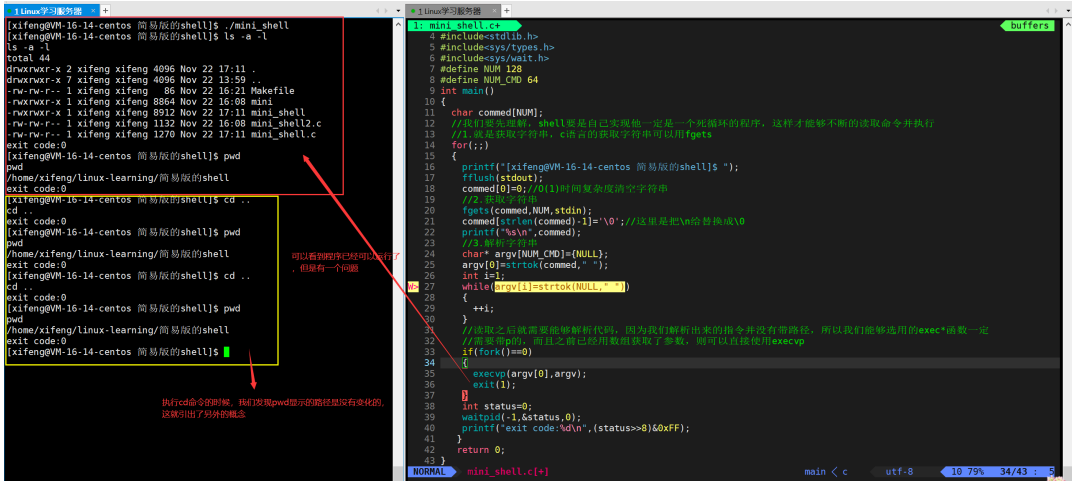
3. 解析命令字符串



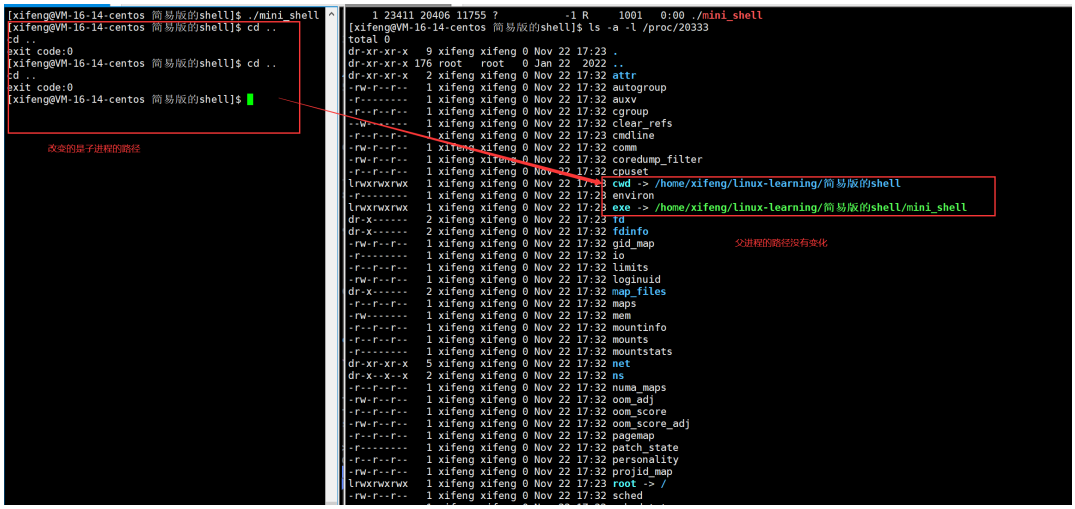
- strtok()

```
STRTOK(3)                Linux Programmer's Manual                STRTOK(3)

NAME
       strtok, strtok_r - extract tokens from strings

SYNOPSIS
       #include <string.h>

       char *strtok(char *str, const char *delim);

       char *strtok_r(char *str, const char *delim, char **saveptr);

   Feature    Test    Macro    Requirements    for    glibc    (see    fea-
   ture_test_macros(7)):

       strtok_r(): _SVID_SOURCE || _BSD_SOURCE || _POSIX_C_SOURCE >= 1
       || _XOPEN_SOURCE || _POSIX_SOURCE

DESCRIPTION
       The strtok() function breaks a string into a sequence of zero or
       more nonempty tokens.  On the first call to strtok() the  string
       to  be  parsed  should  be specified in str.  In each subsequent
       call that should parse the same string, str must be NULL.

       The delim argument specifies a set of  bytes  that  delimit  the
       tokens  in  the parsed string.  The caller may specify different
       strings in delim in successive calls that parse the same string.

       Each call to strtok() returns a  pointer  to  a  null-terminated
       string  containing the next token.  This string does not include
       the delimiting byte.  If no  more  tokens  are  found,  strtok()
       returns NULL.
```

- 解析完成就可以直接创建子进程来运行了



- 出现这种现象的主要原因是我们通过子进程来执行cd命令，子进程是回退了，可是并不会影响父进程，pwd显示的是当前进程的路径

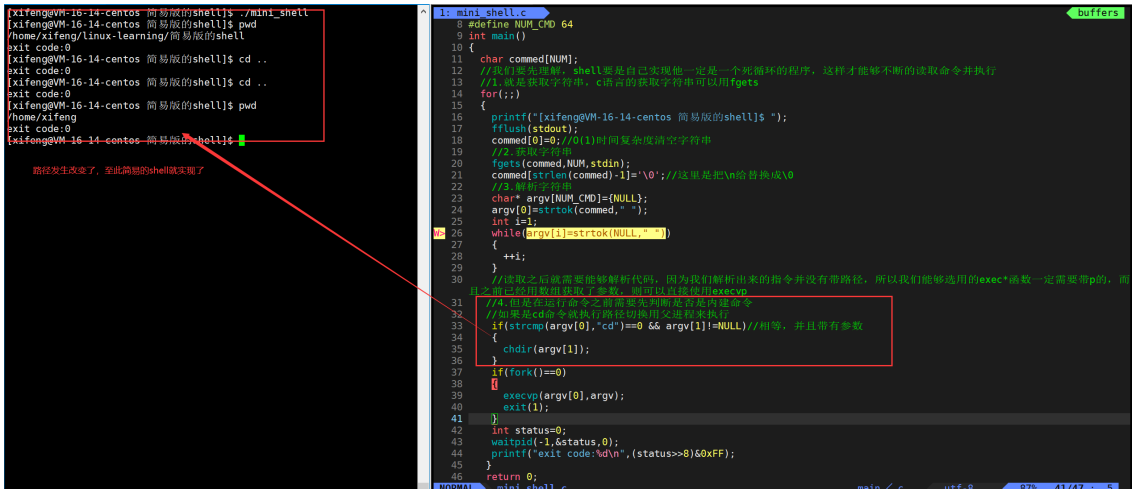


- cd这种命令和ls这种命令不一样，**cd这种属于内建命令**，他是需要父进程来执行的，而不是子进程去执行；其他的属于**第三方命令**，由子进程执行(/usr/bin下面看到的都是第三方命令)
- 所以这个程序就需要步骤4

4. 检测命令是否是需要shell本身执行的，内建命令

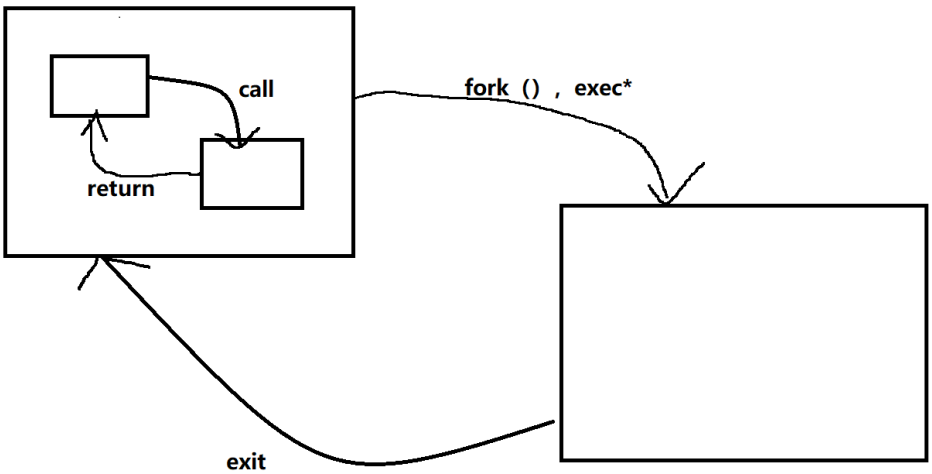
这里就需要一个函数chdir来更改路径

```
CHDIR(2)          Linux Programmer's Manual          CHDIR(2)

NAME
       chdir, fchdir - change working directory

SYNOPSIS
       #include <unistd.h>

       int chdir(const char *path);
       int fchdir(int fd);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       fchdir():
           _BSD_SOURCE || _XOPEN_SOURCE >= 500 ||
           _XOPEN_SOURCE && _XOPEN_SOURCE_EXTENDED
           || /* Since glibc 2.12: */
           _POSIX_C_SOURCE >= 200809L

DESCRIPTION
       chdir() changes the current working directory of the
       calling process to the directory specified in path.

       fchdir() is identical to chdir(); the only difference
       is that the directory is given as an open file
       descriptor.

RETURN VALUE
       On success, zero is returned. On error, -1 is
       returned, and errno is set appropriately.
```



到这里简易的shell就完成了，有一个小问题，思考一下函数和进程之前有没有相似性呢？--->答案是有的

exec/exit就像call/return一个C程序有很多函数组成。一个函数可以调用另外一个函数，同时传递给它一些参数。被调用的函数执行一定的
操作，然后返回一个值。每个函数都有他的局部变量，不同的函数通过call/return系统进行通信。这种通过参数和返回值在拥有私有数据的函数间通信的模式是**结构化程序设计的基础**



一个C程序可以fork/exec另一个程序，并传给它一些参数。这个被调用的程序执行一定的操作，然后通过exit(n)来返回值。调用它的进程可以通过wait来获取exit的返回值。