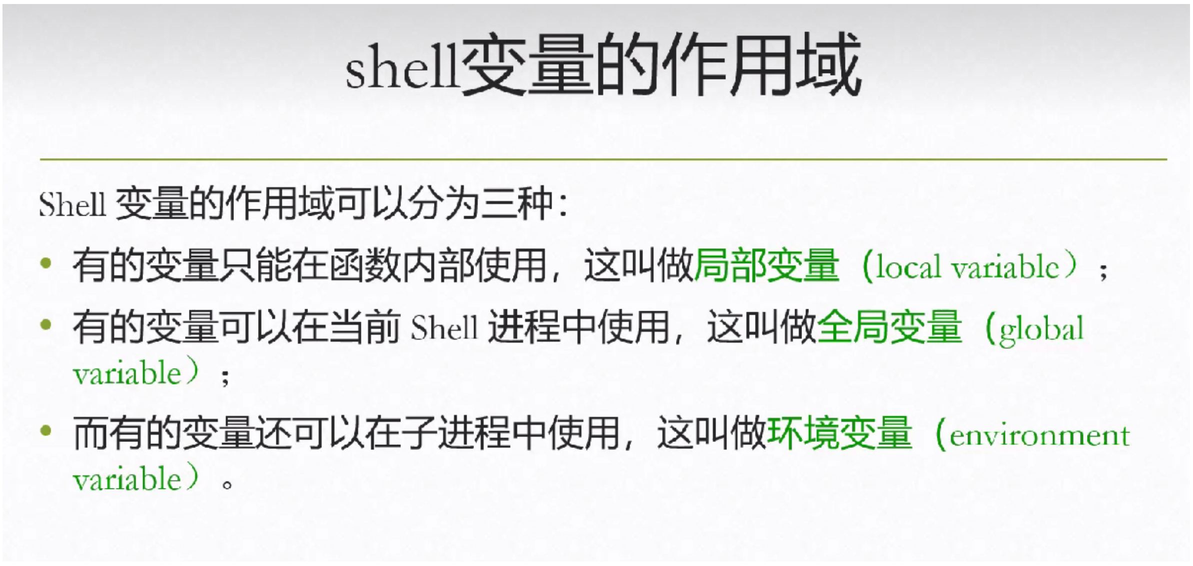
4Shell变量

设置环境变量。

一个shell就是一个进程。

shell作用域



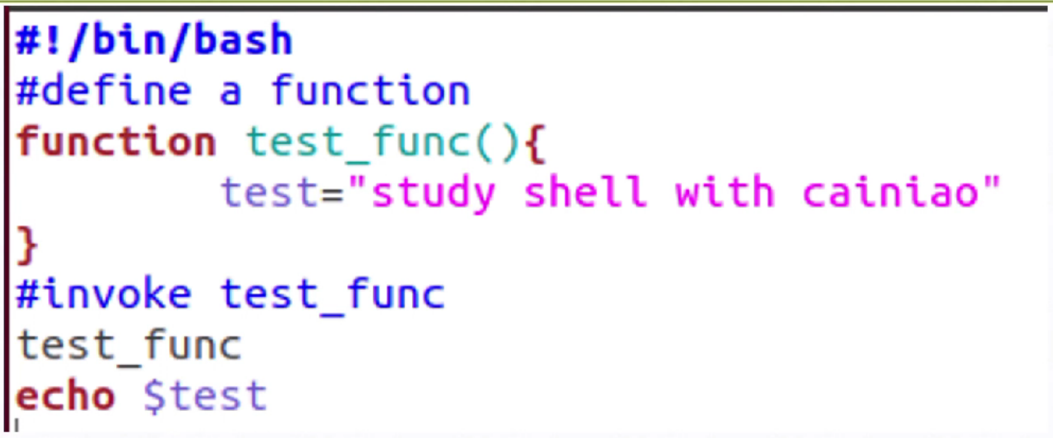
shell变量的作用域

Shell 变量的作用域可以分为三种：

- 有的变量只能在函数内部使用，这叫做局部变量（local variable）；
- 有的变量可以在当前 Shell 进程中使用，这叫做全局变量（global variable）；
- 而有的变量还可以在子进程中使用，这叫做环境变量（environment variable）。

局部变量

Shell 局部变量



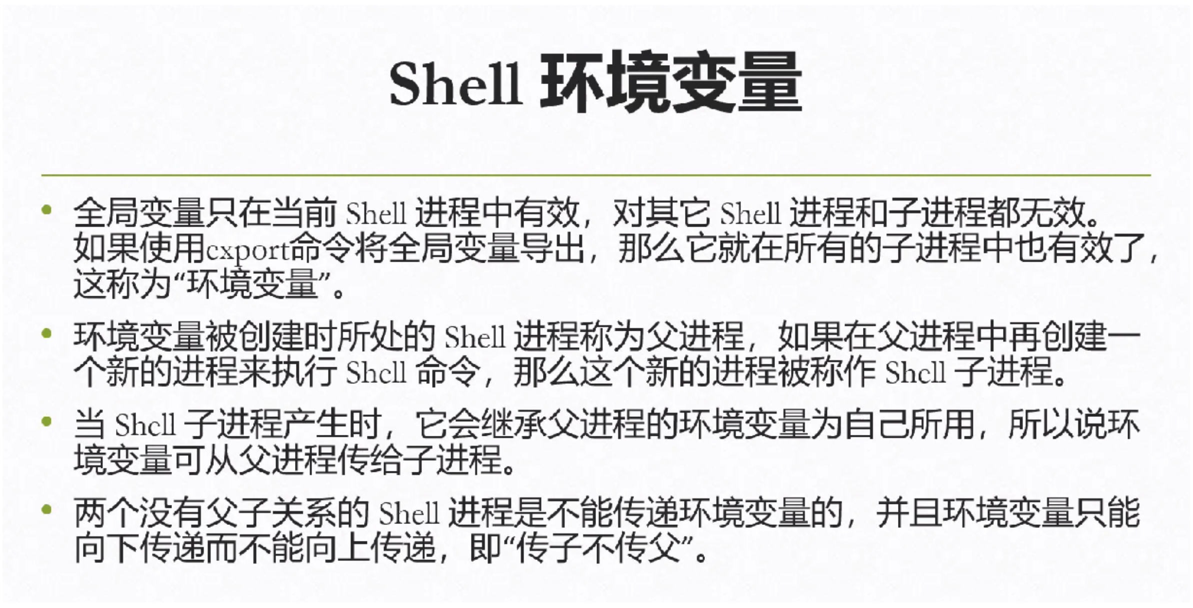
```
#!/bin/bash
#define a function
function test_func(){
        test="study shell with cainiao"
}
#invoke test_func
test_func
echo $test
```

全局变量

Shell 全局变量

- 需要强调的是，全局变量的作用范围是当前的 Shell 进程，而不是当前的 Shell 脚本文件，它们是不同的概念。
- 打开一个 Shell 窗口就创建了一个 Shell 进程，打开多个 Shell 窗口就创建了多个 Shell 进程，每个 Shell 进程都是独立的，拥有不同的进程 ID。
- 在一个 Shell 进程中可以使用 source 命令执行多个 Shell 脚本文件，此时全局变量在这些脚本文件中都有效。

环境变量


Shell 环境变量

- 全局变量只在当前 Shell 进程中有效，对其它 Shell 进程和子进程都无效。如果使用cxport命令将全局变量导出，那么它就在所有的子进程中也有效了，这称为“环境变量”。
- 环境变量被创建时所处的 Shell 进程称为父进程，如果在父进程中再创建一个新的进程来执行 Shcll 命令，那么这个新的进程被称作 Shcll 子进程。
- 当 Shcll 子进程产生时，它会继承父进程的环境变量为自己所用，所以说环境变量可从父进程传给子进程。
- 两个没有父子关系的 Shell 进程是不能传递环境变量的，并且环境变量只能向下传递而不能向上传递，即“传子不传父”。

所以这个题就是什么呢？就是设置环境变量。

直接设置全局变量：

```
ubuntu@VM-12-10-ubuntu[21:53:23]:~
$ basxvv=abcdefg
ubuntu@VM-12-10-ubuntu[21:53:35]:~
$ echo basxvv
basxvv
ubuntu@VM-12-10-ubuntu[21:54:27]:~
$ echo $basxvv
abcdefg
#只在当前shell有效
```

将全局扩展为环境

```
$ export basxvv
#不需要$号
```

5-6|Linux Shell重定向

关于Linux Shell重定向（输入输出重定向）

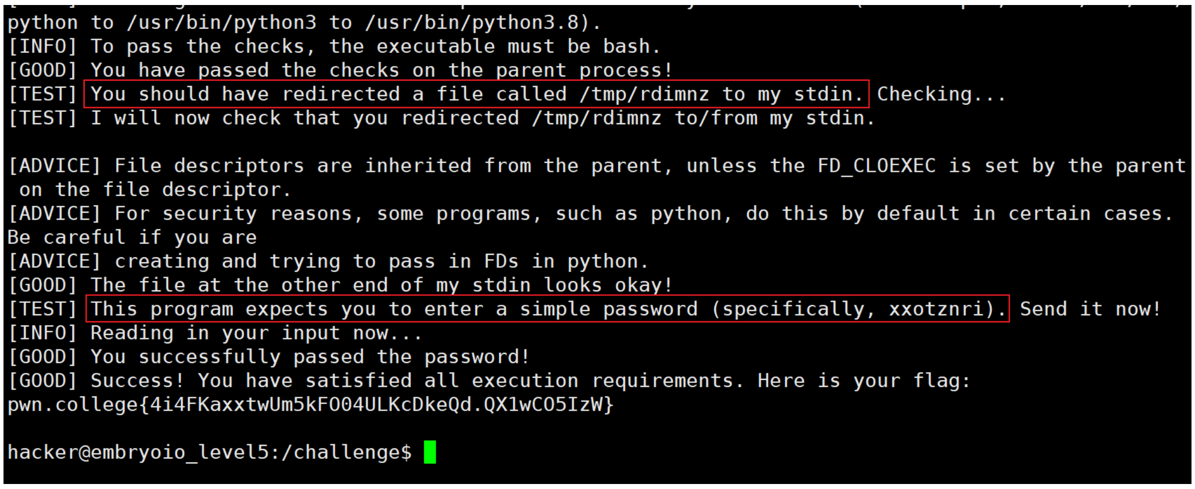
http://c.biancheng.net/view/942.html

https://www.runoob.com/linux/linux-shell-io-redirections.html

exp:

```
./embryoio_level5 < /tmp/rdimnz
#在/tmp/rdimnz写入密码
```

结果如下：



```
python to /usr/bin/python3 to /usr/bin/python3.8).
[INFO] To pass the checks, the executable must be bash.
[GOOD] You have passed the checks on the parent process!
[TEST] You should have redirected a file called /tmp/rdimnz to my stdin. Checking...
[TEST] I will now check that you redirected /tmp/rdimnz to/from my stdin.

[ADVICE] File descriptors are inherited from the parent, unless the FD_CLOEXEC is set by the parent
 on the file descriptor.
[ADVICE] For security reasons, some programs, such as python, do this by default in certain cases.
Be careful if you are
[ADVICE] creating and trying to pass in FDs in python.
[GOOD] The file at the other end of my stdin looks okay!
[TEST] This program expects you to enter a simple password (specifically, xxotznri). Send it now!
[INFO] Reading in your input now...
[GOOD] You successfully passed the password!
[GOOD] Success! You have satisfied all execution requirements. Here is your flag:
pwn.college{4i4FKaxxtwUm5kFO04ULKcDkeQd.QX1wCO5IzW}

hacker@embryoio_level5:/challenge$
```

6

和5相反，是输出：

```
hacker@embryoio_level6:/challenge$
./embryoio_level6 > /tmp/dldsqm
```

然后cat这个文件：flag就在这个文件里面。

7Linux指令:env

第二中方法：

```
exec -c binary
```

第三种方法：

利用C语言函数，exec家族：

```
execve
       #include <unistd.h>

       int execve(const char *pathname, char
*const argv[],
                  char *const envp[]);
```

空env，直接参数改为NULL即可。

```
#include<unistd.h>

int main(int argc,char** argv)
{
    execve(argv[1],NULL,NULL);
}

$/mybinary /binary
```

env:

```
james@james-vm:~/Desktop/pwn.college$ env --h
Usage: env [OPTION]... [-] [NAME=VALUE]...
[COMMAND [ARG]...]
Set each NAME to VALUE in the environment and
run COMMAND.

Mandatory arguments to long options are
mandatory for short options too.
  -i, --ignore-environment  start with an
empty environment
  -0, --null           end each output line
with NUL, not newline
  -u, --unset=NAME     remove variable from
the environment
  -C, --chdir=DIR      change working
directory to DIR
  -S, --split-string=S  process and split S
into separate arguments;
```

```
                        used to pass multiple
arguments on shebang lines
      --block-signal[=SIG]    block delivery
of SIG signal(s) to COMMAND
      --default-signal[=SIG]  reset handling
of SIG signal(s) to the default
      --ignore-signal[=SIG]   set handling of
SIG signals(s) to do nothing
      --list-signal-handling  list non
default signal handling to stderr
  -v, --debug          print verbose
information for each processing step
      --help     display this help and exit
      --version  output version information
and exit

A mere - implies -i.  If no COMMAND, print
the resulting environment.
```

这道题，直接：

```
$env -i ./binary
```

学到了，学到了

对于一个标准的C语言程序，要有：

argc：参数个数，

argv，参数

envp：环境变量

以下程序是统计参数和环境变量的，只是说这个程序在输出环境变量时不能正常运行，会出错。

而环境变量正是上面我们写的shell变量。

```
#include<stdio.h>
int main(int argc,char **argv,char **envp)
{

    printf("The number of arguments is:
%d\n", argc);
    for(int i=0;i<argc;i++)
    {
        printf("The %dth argument is:
%s\n",i+1, argv[i]);
    }
    printf("THe last argument is
%s\n",argv[argc]);
    //printf("THe last argument is
%s\n",argv[argc+1]);
    printf("---------------------------------
------------------\n");
    int num_vars = 0;
    while (*envp)
    {
        envp++;
        num_vars++;
    }

    printf("There are %d environment
variables.\n", num_vars);
    for(int i=0;i<num_vars;i++)
    {
        printf("The %dth environment variable
is: %s\n",i+1, envp[i]);
    }

    int output_num_vars=0;
    /*
    while (*envp)
    {
        printf("The %dth environment variable
is: %s\n",output_num_vars+1,
envp[output_num_vars]);
        envp++;
        output_num_vars++;
    }
    */
}
```

8-15|shell编程

https://www.runoob.com/linux/linux-shell.html

可以在/tmp/文件夹下写一个.sh文件，里面写上：

```
#!/bin/bash
/challenge/embryoio_level8
```

#! 是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种 Shell。

然后执行：

```
$bash /tmp/x.sh
```

9

跟8一样，只是后面输入一个密码。

10

跟8一样，无非加一个参数。

```
#!/bin/bash
/challenge/embryoio_level10 密码
```

11

```
#!/bin/bash
export yzliel=cfiujxkqok
/challenge/embryoio_level11
```

12

1.sh的内容:

```
#!/bin/bash
/challenge/embryoio_level12 < /tmp/oaoqbm
```

/tmp/oaoqbm里面写入密码。

13

```
#!/bin/bash
/challenge/embryoio_level13 > /tmp/xkniyr
```

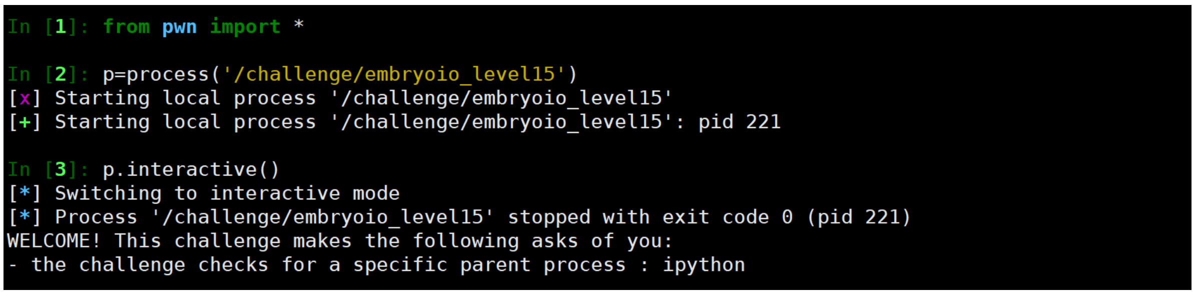
14

空环境变量

```
#!/bin/bash
env -i /challenge/embryoio_level14
```

15ipython

想让你学会ipython


```
In [1]: from pwn import *

In [2]: p=process('/challenge/embryoio_level15')
[x] Starting local process '/challenge/embryoio_level15'
[+] Starting local process '/challenge/embryoio_level15': pid 221

In [3]: p.interactive()
[*] Switching to interactive mode
[*] Process '/challenge/embryoio_level15' stopped with exit code 0 (pid 221)
WELCOME! This challenge makes the following asks of you:
- the challenge checks for a specific parent process : ipython
```

p.interactive()

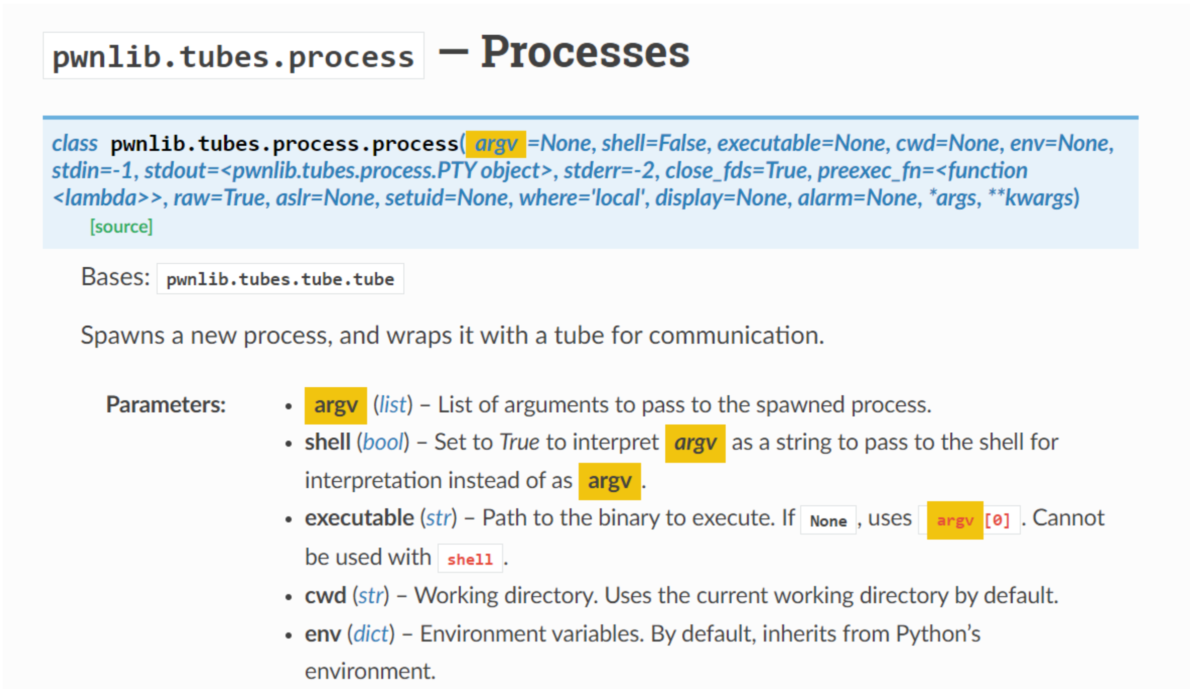
或者p.read()都可

16

在15的基础上，输入个密码就行了

17-28|process

看pwntools:


pwnlib.tubes.process — Processes

class **pwnlib.tubes.process.process**(*argv=None, shell=False, executable=None, cwd=None, env=None, stdin=-1, stdout=<pwnlib.tubes.process.PTY object>, stderr=-2, close_fds=True, preexec_fn=<function <lambda>>, raw=True, aslr=None, setuid=None, where='local', display=None, alarm=None, *args, **kwargs*) [source]

Bases: `pwnlib.tubes.tube.tube`

Spawns a new process, and wraps it with a tube for communication.

Parameters:
- **argv** (*list*) – List of arguments to pass to the spawned process.
- **shell** (*bool*) – Set to *True* to interpret *argv* as a string to pass to the shell for interpretation instead of as argv.
- **executable** (*str*) – Path to the binary to execute. If `None`, uses `argv[0]`. Cannot be used with `shell`.
- **cwd** (*str*) – Working directory. Uses the current working directory by default.
- **env** (*dict*) – Environment variables. By default, inherits from Python's environment.

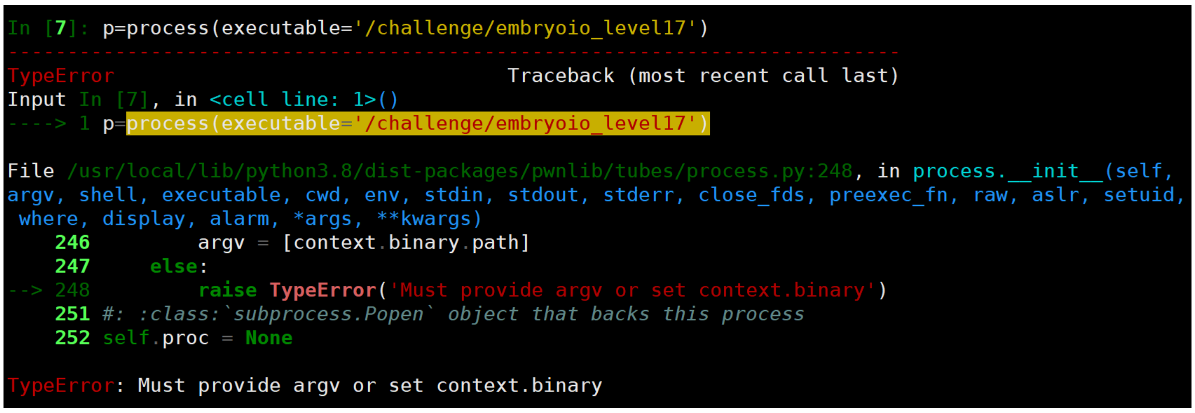
看源码吧：

```
"""
        argv(list):
            List of arguments to pass to the
spawned process.
        shell(bool):
            Set to `True` to interpret `argv`
as a string
            to pass to the shell for
interpretation instead of as argv.
        executable(str):
            Path to the binary to execute.
If :const:`None`, uses ``argv[0]``.
            Cannot be used with ``shell``.
        cwd(str):
            Working directory.  Uses the
current working directory by default.
        env(dict):
            Environment variables.  By
default, inherits from Python's environment.
        .......
"""



def __init__(self, argv = None,
                 shell = False,
                 executable = None,
                 cwd = None,
                 env = None,
                 stdin  = PIPE,
                 stdout = PTY,
                 stderr = STDOUT,
                 close_fds = True,
                 preexec_fn = lambda: None,
                 raw = True,
                 aslr = None,
                 setuid = None,
                 where = 'local',
                 display = None,
                 alarm = None,
                 *args,
                 **kwargs
                 ):
        super(process,
self).__init__(*args,**kwargs)

        # Permit using context.binary
        if argv is None:
            if context.binary:
                argv = [context.binary.path]
            else:
                raise TypeError('Must provide
argv or set context.binary')

            ......
```

也就是说，关键是参数，也就是argv。看下面的代码，没有参数，就无法执行process。

```
  # Permit using context.binary
          if argv is None:
              if context.binary:
                  argv = [context.binary.path]
              else:
                  raise TypeError('Must provide
 argv or set context.binary')
```

示例：



```
In [7]: p=process(executable='/challenge/embryoio_level17')
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [7], in <cell line: 1>()
----> 1 p=process(executable='/challenge/embryoio_level17')

File /usr/local/lib/python3.8/dist-packages/pwnlib/tubes/process.py:248, in process.__init__(self,
argv, shell, executable, cwd, env, stdin, stdout, stderr, close_fds, preexec_fn, raw, aslr, setuid,
 where, display, alarm, *args, **kwargs)
    246         argv = [context.binary.path]
    247     else:
--> 248         raise TypeError('Must provide argv or set context.binary')
    251 #: :class:`subprocess.Popen` object that backs this process
    252 self.proc = None

TypeError: Must provide argv or set context.binary
```

出错部分的代码正是上面那一段。

所谓的binary，也是参数，argv[0]。

```
p=process("./binary")
'''
显然，只有一个参数，也就是argv
argv是list,显然只有一个值，也就是argv[0]
一个程序的执行，我感觉也是向某个*程序管理程序*传
入参数，让它执行。
当然，参数argv[0]就是想要执行的程序的路径，
argv[1]...就是传给要执行
的程序的参数了。
'''
```

如果没有参数，没有明确了executable，不执行。

如果有参数，没有明确了executable，那么executable = argv[0]

也就是将argv[0]作为执行文件的路径。

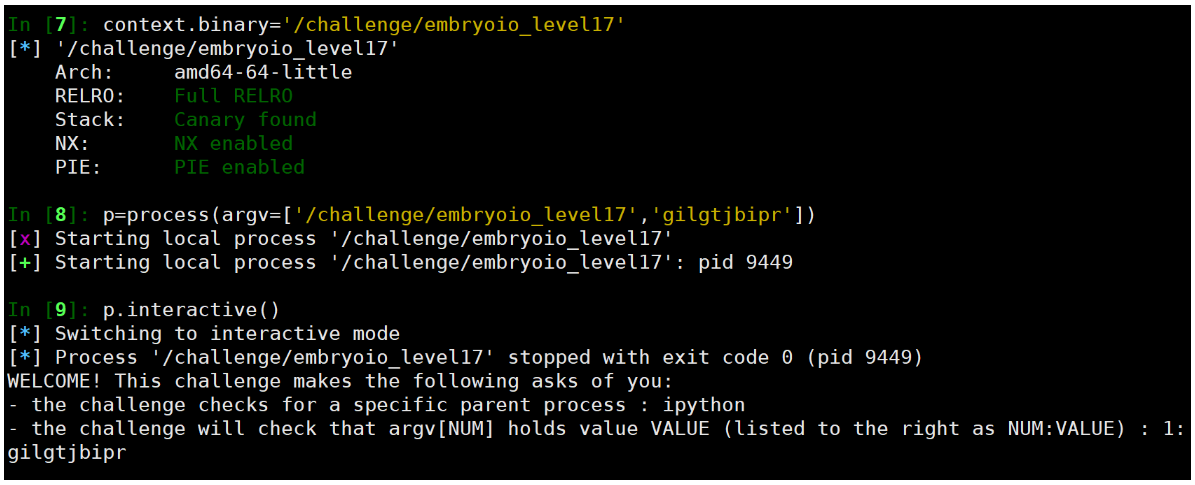
```
        if not executable:
            if not argv:
                self.error("Must specify argv
or executable")
            executable = argv[0]
```

思考：那么如何给想要执行的程序传递参数呢？

argv一定要有，目前也没找到什么别的办法。

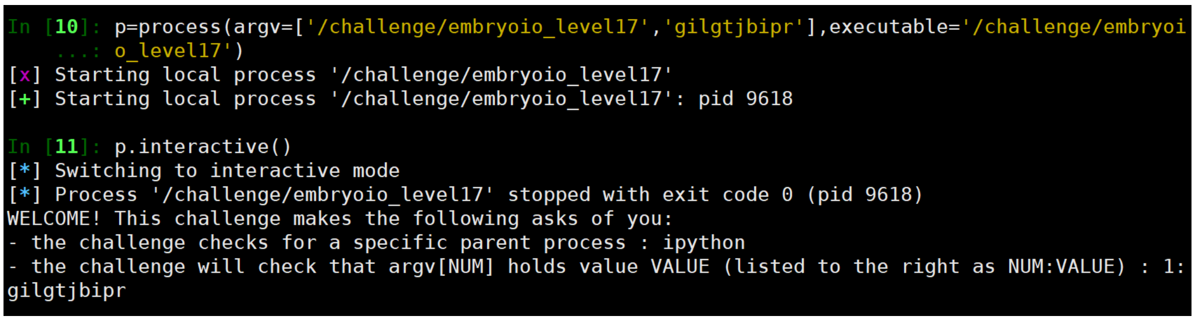
所以就直接用argv，作为list来进行传参，只有后面你加不加context.binary或者executable其实无所谓，不影响。

picture1:



```
In [7]: context.binary='/challenge/embryoio_level17'
[*] '/challenge/embryoio_level17'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled

In [8]: p=process(argv=['/challenge/embryoio_level17','gilgtjbipr'])
[x] Starting local process '/challenge/embryoio_level17'
[+] Starting local process '/challenge/embryoio_level17': pid 9449

In [9]: p.interactive()
[*] Switching to interactive mode
[*] Process '/challenge/embryoio_level17' stopped with exit code 0 (pid 9449)
WELCOME! This challenge makes the following asks of you:
- the challenge checks for a specific parent process : ipython
- the challenge will check that argv[NUM] holds value VALUE (listed to the right as NUM:VALUE) : 1:
gilgtjbipr
```

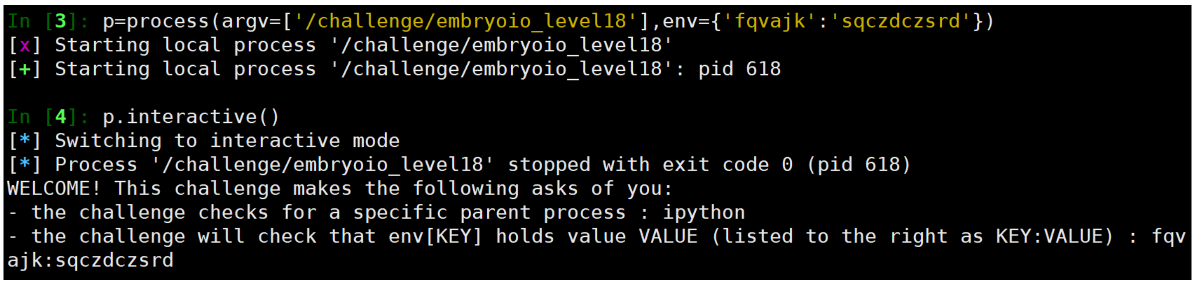
picture2:



```
In [10]: p=process(argv=['/challenge/embryoio_level17','gilgtjbipr'],executable='/challenge/embryoi
    ...: o_level17')
[x] Starting local process '/challenge/embryoio_level17'
[+] Starting local process '/challenge/embryoio_level17': pid 9618

In [11]: p.interactive()
[*] Switching to interactive mode
[*] Process '/challenge/embryoio_level17' stopped with exit code 0 (pid 9618)
WELCOME! This challenge makes the following asks of you:
- the challenge checks for a specific parent process : ipython
- the challenge will check that argv[NUM] holds value VALUE (listed to the right as NUM:VALUE) : 1:
gilgtjbipr
```

18

无非是使用env



```
In [3]: p=process(argv=['/challenge/embryoio_level18'],env={'fqvajk':'sqczdczsrd'})
[x] Starting local process '/challenge/embryoio_level18'
[+] Starting local process '/challenge/embryoio_level18': pid 618

In [4]: p.interactive()
[*] Switching to interactive mode
[*] Process '/challenge/embryoio_level18' stopped with exit code 0 (pid 618)
WELCOME! This challenge makes the following asks of you:
- the challenge checks for a specific parent process : ipython
- the challenge will check that env[KEY] holds value VALUE (listed to the right as KEY:VALUE) : fqv
ajk:sqczdczsrd
```

env要求是dic，也就是字典。

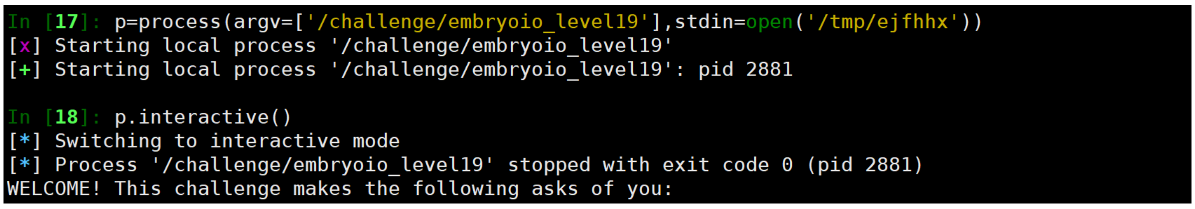
字典实例：

```
d = {key1 : value1, key2 : value2 }
```

19

对于process，设计者想让你有更为深刻的理解。

这是在discord上抄的：



```
In [17]: p=process(argv=['/challenge/embryoio_level19'],stdin=open('/tmp/ejfhhx'))
[x] Starting local process '/challenge/embryoio_level19'
[+] Starting local process '/challenge/embryoio_level19': pid 2881

In [18]: p.interactive()
[*] Switching to interactive mode
[*] Process '/challenge/embryoio_level19' stopped with exit code 0 (pid 2881)
WELCOME! This challenge makes the following asks of you:
```

深夜了，要睡觉了。

无论做什么，都不应当损害自己的身体，尤其是眼睛。

我们注意如下：

```
                    stdin  = PIPE,
                    stdout = PTY,
                    stderr = STDOUT,
```

PTY是一个定义的对象，没有实际含义。PTY是什么？其实可以理解为就是终端。/dev/tty就是这样。

PIPE就是管道。可以理解为FIFO对象。

其实这二者，都可以使用普通的文件流即可。

20

Python open() 函数

```
open(name[, mode[, buffering]])
```

- mode : mode 决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。

所以最好强调使用文件是在那种mdoe下。

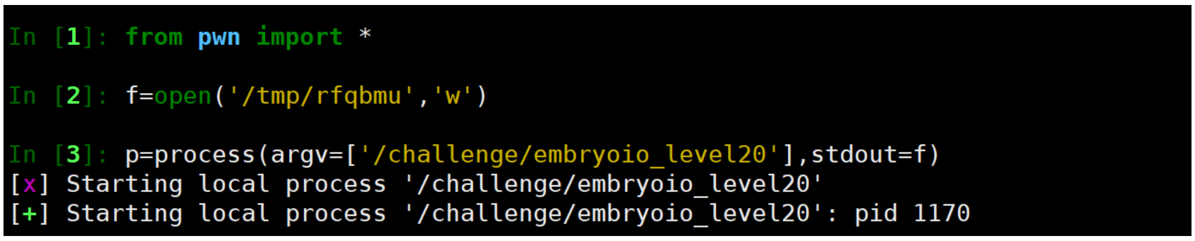
Linux shell中的竖线（|）——管道符号

管道符号，是unix一个很强大的功能,符号为一条竖线:"|"。
用法：command 1 | command 2 他的功能是把第一个命令command 1执行的结果作为command 2的输入传给command 2

其实很简单，使用IO流即可。

所以应该好好学一下python文件输入输出流。

截图：



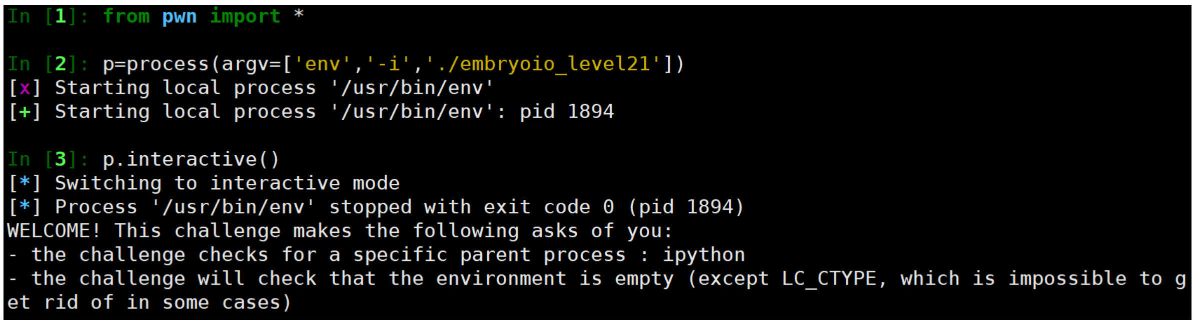
```
In [1]: from pwn import *

In [2]: f=open('/tmp/rfqbmu','w')

In [3]: p=process(argv=['/challenge/embryoio_level20'],stdout=f)
[x] Starting local process '/challenge/embryoio_level20'
[+] Starting local process '/challenge/embryoio_level20': pid 1170
```

21

这个简单，无非是在process里面执行

```
$env -i binary
```

截图：



```
In [1]: from pwn import *

In [2]: p=process(argv=['env','-i','./embryoio_level21'])
[x] Starting local process '/usr/bin/env'
[+] Starting local process '/usr/bin/env': pid 1894

In [3]: p.interactive()
[*] Switching to interactive mode
[*] Process '/usr/bin/env' stopped with exit code 0 (pid 1894)
WELCOME! This challenge makes the following asks of you:
- the challenge checks for a specific parent process : ipython
- the challenge will check that the environment is empty (except LC_CTYPE, which is impossible to g
et rid of in some cases)
```

22

```
from pwn import *
p=process('/challenge/embryoio_level22')
p.interactive()
```

23

在22基础上，输入个密码

24

```python
from pwn import *
p=process(['/challenge/embryoio_level22','fcpufvxkna'])
p.interactive()
```

25

```python
from pwn import *
p=process(['/challenge/embryoio_level25'],env={'fabrlj':'vcjrvnsvzp'})
p.interactive()
```

26

```python
from pwn import *
f=open('/tmp/otxeyo')
p=process(['/challenge/embryoio_level26'],stdin=f)
p.interactive()
```

27

```
from pwn import *
f=open('/tmp/bvlhtg','w')
p=process('/challenge/embryoio_level27',stdou
t=f)
p.interactive()
```

28

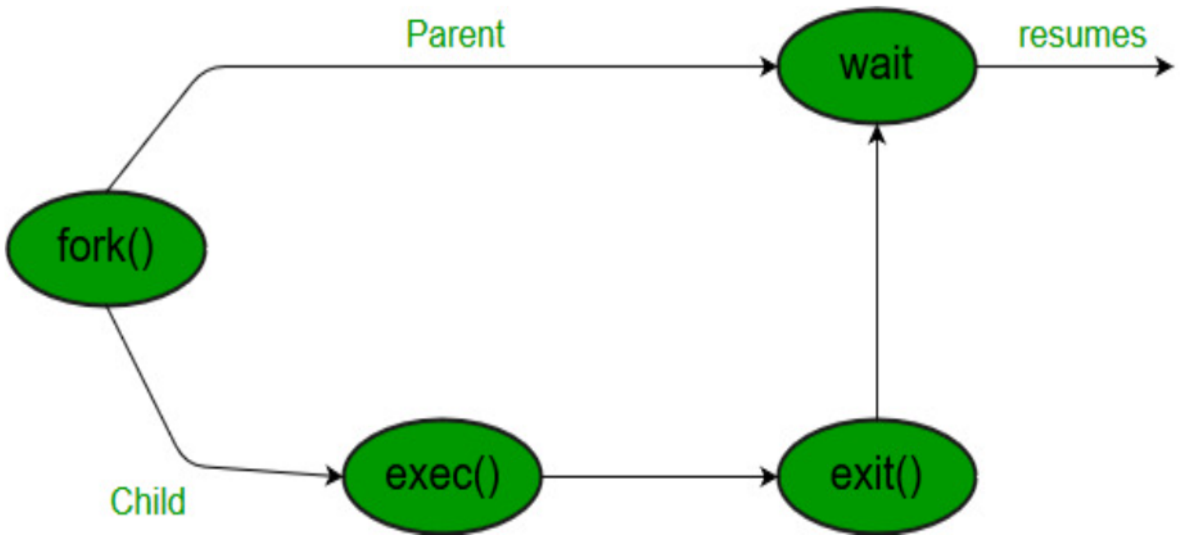
```
from pwn import *
p=process(['env','-
i','/challenge/embryoio_level'])
p.interactive()
```

29

开始难起来了。

`linux`

一个很重要的截图:



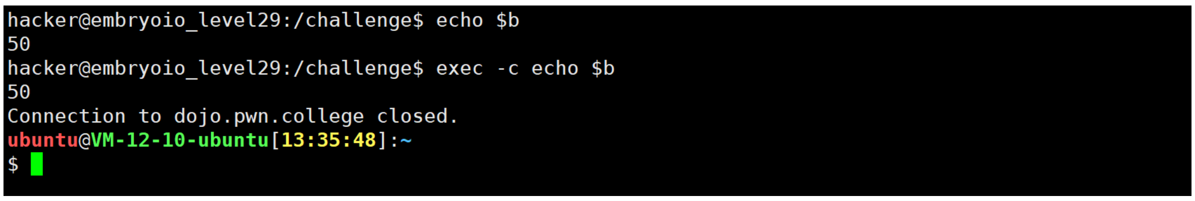
关于shell和终端

https://www.ihewro.com/archives/933/

终端，里面有各种各样的shell。

当前shell，然后能够产生子shell

使用exec指令，会直接用command替换当前进程，也就是当前shell进程被替换，就直接退出shell。（待研究）



```
hacker@embryoio_level29:/challenge$ echo $b
50
hacker@embryoio_level29:/challenge$ exec -c echo $b
50
Connection to dojo.pwn.college closed.
ubuntu@VM-12-10-ubuntu[13:35:48]:~
$ 
```

fork()函数

还是很有趣的，fork函数。

生成一个子进程，子进程和父进程执行一样的程序。

函数原型：

```
int fock();
//暂无资料
```

返回值：

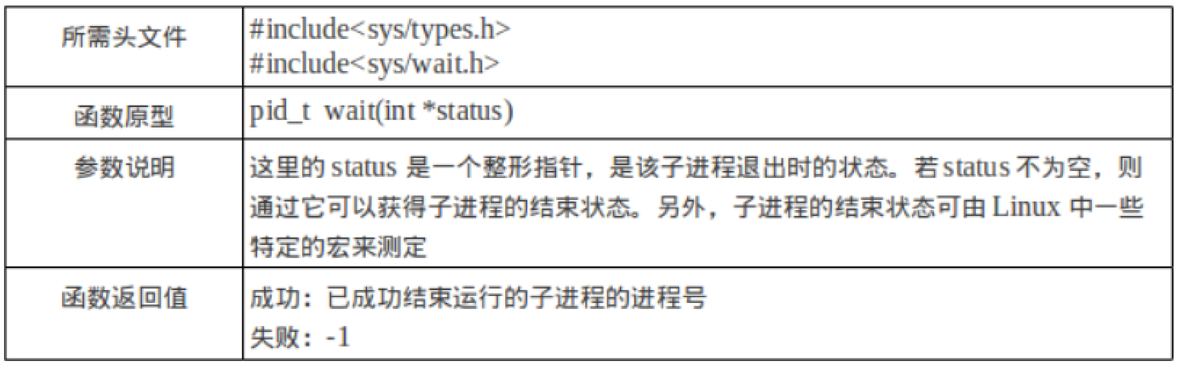
> *Negative Value*: creation of a child process was unsuccessful.
>
> 负值：创建失败
>
> *Zero*: Returned to the newly created child process.
>
> 0：新创建的进程
>
> *Positive value*: Returned to parent or caller. The value contains process ID of newly created child process.
>
> 正值：新创建的子进程的id

wait()函数

函数原型：

```
// take one argument status and returns
// a process ID of dead children.
pid_t wait(int *status);
```

截图：



所需头文件	#include<sys/types.h>
	#include<sys/wait.h>

函数原型	pid_t wait(int *status)
参数说明	这里的 status 是一个整形指针，是该子进程退出时的状态。若 status 不为空，则通过它可以获得子进程的结束状态。另外，子进程的结束状态可由 Linux 中一些特定的宏来测定
函数返回值	成功：已成功结束运行的子进程的进程号 失败：-1

waitpid()函数

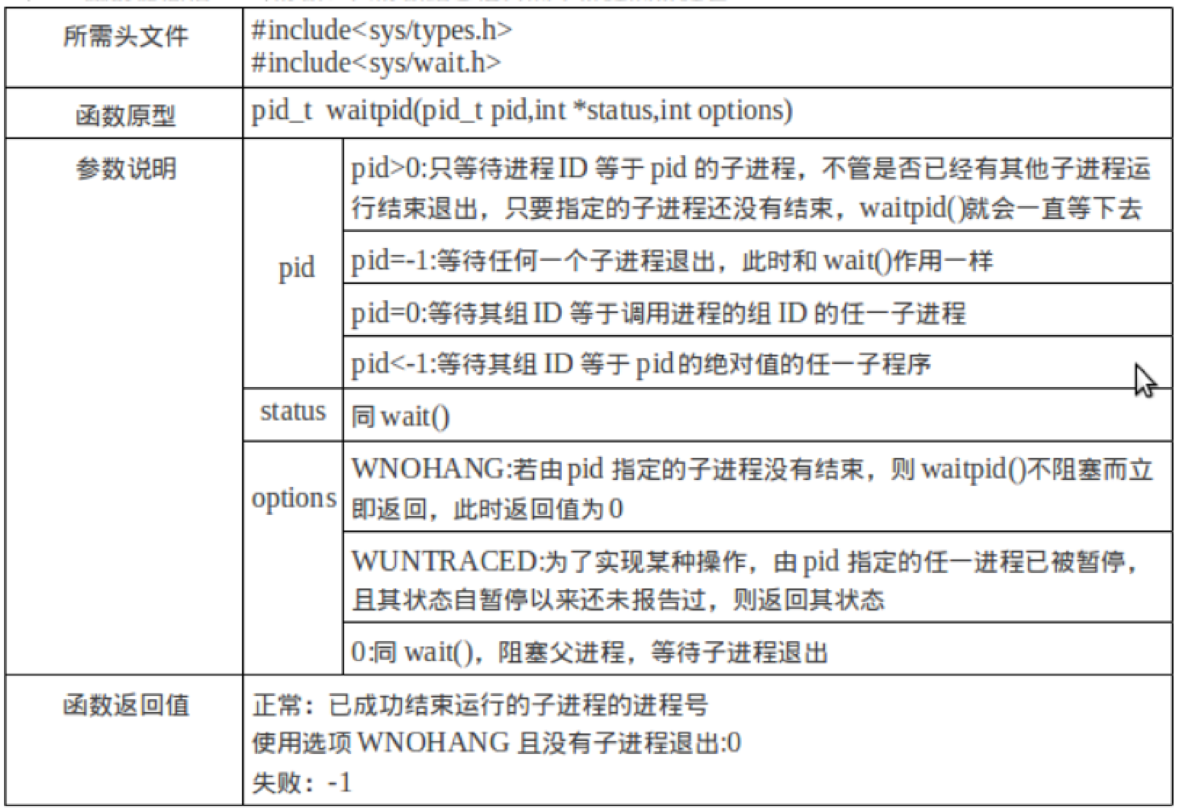
函数原型：

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid,int *status,int
options);
```

参数解析：

- pid是想要等待的子进程的pid
- status是
- options是可选项

截图：



所需头文件	#include<sys/types.h>
	#include<sys/wait.h>	

函数原型	pid_t waitpid(pid_t pid,int *status,int options)	
参数说明	pid	pid>0:只等待进程ID等于pid的子进程，不管是否已经有其他子进程运行结束退出，只要指定的子进程还没有结束，waitpid()就会一直等下去
		pid=-1:等待任何一个子进程退出，此时和wait()作用一样
		pid=0:等待其组ID等于调用进程的组ID的任一子进程
		pid<-1:等待其组ID等于pid的绝对值的任一子程序
	status	同wait()
	options	WNOHANG:若由pid指定的子进程没有结束，则waitpid()不阻塞而立即返回，此时返回值为0
		WUNTRACED:为了实现某种操作，由pid指定的任一进程已被暂停，且其状态自暂停以来还未报告过，则返回其状态
		0:同wait()，阻塞父进程，等待子进程退出
函数返回值	正常：已成功结束运行的子进程的进程号 使用选项WNOHANG且没有子进程退出:0 失败：-1	

exec Family

太多了。

> The *const char *arg* and subsequent ellipses in the **execl**(), **execlp**(), and **execle**() functions can be thought of as *arg0*, *arg1*, ..., *argn*. Together they describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program. **The first argument,**

> **by convention, should point to the filename associated with the file being executed.** ***The list of arguments must be terminated by a NULL pointer,*** and, since these are variadic functions, this pointer must be cast *(char *) NULL*.

注意加粗部分：

第一个参数，也就是argv[0]，一般都是要执行的filename。

当然，那你其实多少有点多余了。第一个参数filename传入了，结果参数又要传入一个。

最后一个参数必须是NULL。

注意：使用exec系列函数执行时不会改变新进程的父亲，相当于只是将当前进程替换掉了。

exec家族函数需要的参数，基本上都是const限定的常量。

```
#include <unistd.h>

extern char **environ;
```

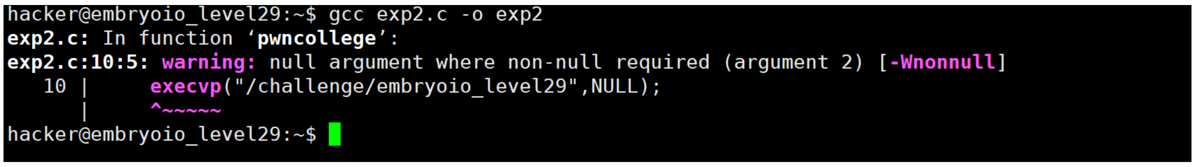
```
int execl(const char *path, const char *arg,
...);

int execlp(const char *file, const char *arg,
...);

int execle(const char *path, const char *arg,
..., char * const envp[]);

int execv(const char *path, char *const
argv[]);

int execvp(const char *file, char *const
argv[]);

int execvpe(const char *file, char *const
argv[],
char *const envp[]);
```

看需要使用吧，有的可以设置环境变量。

exp

注意：

execvp函数，第二个参数为空会被警告。非null参数，却填入了一个null。


```
hacker@embryoio_level29:~$ gcc exp2.c -o exp2
exp2.c: In function ‘pwncollege’:
exp2.c:10:5: warning: null argument where non-null required (argument 2) [-Wnonnull]
   10 |     execvp("/challenge/embryoio_level29",NULL);
      |     ^~~~~~
hacker@embryoio_level29:~$ 
```

waitpid的第三个参数为0，就和wait一样了。

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>

void pwncollege() {
    //printf("pwn.college");
    //int execvp (const char *file, char
*const argv[]);
    char *args=
{"/challenge/embryoio_level29",NULL};


 execvp("/challenge/embryoio_level29",NULL);
    //or
    //这样做不会被警告

 //execvp("/challenge/embryoio_level29",args)
;
    //printf("pwn.college");
}


int main(int argc,char** argv[],char* env[])
{
```

```
    int stat;
    int childPid;
    pid_t pid;

    if (fork() == 0) {
        childPid = getpid();
        printf("child pid is %d\n",
childPid);

        pwncollege();
        printf("ok,i am child ,i will
leave\n");

    } else {
        printf("parent pid is %d\n",
getpid());
        waitpid(childPid,NULL,0);
        printf("ok,i am father ,i will
leave\n");
        return 0;
    }
}
```

30-32

const

const 和指针

const 也可以和指针变量一起使用，这样可以限制指针变量本身，也可以限制指针指向的数据。const 和指针一起使用会有几种不同的顺序，如下所示：

```
const int *p1;
int const *p2;
int * const p3;
```

在最后一种情况下，指针是只读的，也就是 p3 本身的值不能被修改；在前面两种情况下，指针所指向的数据是只读的，也就是 p1、p2 本身的值可以修改（指向不同的数据），但它们指向的数据不能被修改。

当然，指针本身和它指向的数据都有可能是只读的，下面的两种写法能够做到这一点：

```
const int * const p4;
int const * const p5;
```

> const 和指针结合的写法多少有点让初学者摸不着头脑，大家可以这样来记忆：const 离变量名近就是用来修饰指针变量的，离变量名远就是用来修饰指针指向的数据，如果近的和远的都有，那么就同时修饰指针变量以及它指向的数据。

实例：

```
   /* i is stored in read only area*/
    int const i = 10;

    /* pointer to integer constant. Here i(上面定义的i)
    is of type "const int", and &i is of
    type "const int *".  And p is of type
    "const int", types are matching no issue
*/
    int const *ptr = &i;
```

这里说的很对，i是一个整形常量。

那么：

地址	值	类型
&i	i	const int*
i	10(也就是一个立即数)	const int

exp

我对于exec函数需要的参数，全部使用函数要求的参数，也就是全部使用它规定要求传入的参数数据类型，这样在编译时就不会报错。

至于说参数本身有什么意义？其实没啥意义，尤其是argv，主要是不能为NULL。

```
void pwncollege() {
    //printf("pwn.college");
    //int execvp (const char *file, char
*const argv[]);
    char* const argv[]=
{"/challenge/embryoio_level30",NULL};
    const char*
file="/challenge/embryoio_level30";
    execvp(file,argv);
}


int main(int argc,char** argv[],char* env[])
{
    int childPid;
    pid_t pid;

    if (fork() == 0) {
        childPid = getpid();
        printf("child pid is %d\n",
childPid);

        pwncollege();

        printf("ok,i am child ,i will
leave\n");

    } else {
        printf("parent pid is %d\n",
getpid());

        waitpid(childPid,NULL,0);

        printf("ok,i am father ,i will
leave\n");
        return 0;
    }
}
```

31

需要参数了。

> WELCOME! This challenge makes the following asks of you:

> - the challenge checks for a specific parent process : binary
> - the challenge will check that argv[NUM] holds value VALUE (listed to the right as NUM:VALUE) : 1:ehnyglogio

exp

```
void pwncollege() {
    //注意最后的NULL
    char* const argv[]=
{"/challenge/embryoio_level31","ehnyglogio",NULL};
    const char*
file="/challenge/embryoio_level31";
    execvp(file,argv);
}



int main(int argc,char** argv[],char* env[])
{
    int childPid;
    pid_t pid;

    if (fork() == 0) {
        childPid = getpid();
        printf("child pid is %d\n",
childPid);

        pwncollege();

        printf("ok,i am child ,i will
leave\n");

    } else {
        printf("parent pid is %d\n",
getpid());

        waitpid(childPid,NULL,0);

        printf("ok,i am father ,i will
leave\n");
        return 0;
    }
}
```

32

> WELCOME! This challenge makes the following asks of you:
>
> - the challenge checks for a specific parent process : binary

> - the challenge will check that env[KEY] holds value VALUE (listed to the right as KEY:VALUE) : bljdlc:ynugcmanvu
>   用这个吧:

```
int execvpe(const char *file, char *const
argv[],
char *const envp[]);
```

exp:

注意第一句话：#define _GNU_SOURCE，防止出现函数隐形声明的警告。

但是暂时还没搞清楚为什么，隐式声明还真是万恶之源。

```
#define _GNU_SOURCE
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>

void pwncollege() {
    //注意最后的NULL
    char* const argv[]=
{"/challenge/embryoio_level32",NULL};
```

```
    const char*
file="/challenge/embryoio_level32";
    char *const envp[]={"bljdlc=ynugcmanvu"};
    execvpe(file,argv,envp);
}


int main(int argc,char** argv[],char* env[])
{
    int childPid;
    pid_t pid;

    if (fork() == 0) {
        childPid = getpid();
        printf("child pid is %d\n",
childPid);

        pwncollege();

        printf("ok,i am child ,i will
leave\n");

    } else {
        printf("parent pid is %d\n",
getpid());

        waitpid(childPid,NULL,0);

        printf("ok,i am father ,i will
leave\n");
```

```
            return 0;
        }
    }
```

33-35输入输出重定向

输入重定向

就是一般来讲输入（0）是从键盘输入，但现在输入不再是从键盘输入。

输出重定向

一般来讲，输出（1）是输出到显示器，但现在不再输出到显示器。

> WELCOME! This challenge makes the following asks of you:
>
> - the challenge checks for a specific parent process : binary
> - the challenge will check that input is redirected from a specific file path : /tmp/wjekgt
> - the challenge will check for a hardcoded password over

> stdin : uaynliwq

Function:open()

```
Syntax in C language
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>
int open (const char* Path, int flags [, int
mode ]);
```

> **Parameters**
>
> - Path:
>
>   path to file which you want to use
>
>   - use absolute path begin with “/”, when you are not work in same directory of file.
>   - Use relative path which is only file name with extension, when you are work in same directory of file.
> - flags :多个flags同时使用使用“|”

How you like to use

- **O_RDONLY**: read only,
- **O_WRONLY**: write only,
- **O_RDWR**: read and write,
- **O_CREAT**: create file if it doesn’t exist,
- **O_EXCL**: prevent creation if it already exists
- O_CREAT 与O_EXCL 同时设置，并且欲打开的文件为符号连接，则会打开文件失败.
  O_NOCTTY 如果欲打开的文件为终端机设备时，则不会将该终端机当成进程控制终端机.
  O_TRUNC 若文件存在并且以可写的方式打开时，此旗标会令文件长度清为0，而原来存于该文件的资料也会消失.
  O_APPEND 当读写文件时会从文件尾开始移动，也就是所写入的数据会以附加的方式加入到文件后面.
  O_NONBLOCK 以不可阻断的方式打开文件，也就是无论有无数据读取或等待，都会立即返回进程之中.

O_NDELAY 同O_NONBLOCK.
O_SYNC 以同步的方式打开文件.
O_NOFOLLOW 如果参数 pathname 所指的文件为一符号连接，则会令打开文件失败.
O_DIRECTORY 如果参数 pathname 所指的文件并非为一目录，则会令打开文件失败。

- mode：只有在建立新文件时才会生效，并且会受**umask**影响，所以建立的文件权限为（**mode-umaks**）

    - S_IRWXU00700 权限，代表该文件所有者具有可读、可写及可执行的权限.
      S_IRUSR 或S_IREAD，00400 权限，代表该文件所有者具有可读取的权限.
      S_IWUSR 或S_IWRITE，00200 权限，代表该文件所有者具有可写入的权限.
      S_IXUSR 或S_IEXEC，00100 权限，代表该文件所有者具有可执行的权限.
      S_IRWXG 00070 权限，代表该文件用户组具有可读、可写及可

> 执行的权限.
>
> S_IRGRP 00040 权限，代表该文件用户组具有可读的权限.
>
> S_IWGRP 00020 权限，代表该文件用户组具有可写入的权限.
>
> S_IXGRP 00010 权限，代表该文件用户组具有可执行的权限.
>
> S_IRWXO 00007 权限，代表其他用户具有可读、可写及可执行的权限.
>
> S_IROTH 00004 权限，代表其他用户具有可读的权限
>
> S_IWOTH 00002 权限，代表其他用户具有可写入的权限.
>
> S_IXOTH 00001 权限，代表其他用户具有可执行的权限.
>
> return:
>
> 1、返回值是一个整数。
>
> 2、打开文件成功，返回文件描述符。
>
> 3、打开文件失败，返回-1。

Function:write()

原型：

```
#include <fcntl.h>
size_t write (int fd, void* buf, size_t cnt);
```

Three "System File Tables"

Function:dup() and dup2()

```
int dup(int oldfd);
//oldfd: old file descriptor whose copy is to be created.
//创建一个当前FD的副本，是一个新的整数，比如旧文件是3，这个副本就是4，
//但是FD3,4指向同样的文件
```

作用就是生成fd副本。

```
int dup2(int oldfd, int newfd);
oldfd: old file descriptor
newfd: new file descriptor which is used by
    dup2() to create a copy.
```

作用是什么？是将newfd的内容重定向到oldfd中。

示例1:

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

int main() {
    int old_fd = open("11.txt",
 O_WRONLY|O_APPEND);
    printf("old_fd=%d\n", old_fd);
    if (old_fd < 0) {
        printf("Error\n");
    }
    int new_fd = dup2(old_fd, 1);

 printf("aaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbb
aaaaaaa\n");
}
```

结果是什么？

程序描述符1，也就是stdout的内容，也就是new_fd，被重定向到old_fd。

输出的aaabbb都被写入到11.txt中。

示例2：（太麻烦了，不想写了）

```
```

大致写一下吧，输入重定向就是：

一个文件t作为输入源，然后一个可执行程序b需要输入，输入不是从键盘，而是从t到b。

输出重定向就是：

一个可执行程序b有输出，一个文本文件t接收b的输出，输出不是显示器了，而是从t->b。

exp

```
#define _GNU_SOURCE
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>

void pwncollege() {
    //注意最后的NULL
    char* const argv[]=
{"/challenge/embryoio_level33",NULL};
    const char*
file="/challenge/embryoio_level33";
    //char *const envp[]=
{"bljdlc":"ynugcmanvu"};
    execvp(file,argv);
```

```c
}


int main(int argc,char** argv[],char* env[])
{
    int childPid;

    int old_fd=open("/tmp/wjekgt",O_RDONLY);
    dup2(old_fd,0);

    if (fork() == 0) {
        childPid = getpid();
        printf("child pid is %d\n",
childPid);

        pwncollege();

        printf("ok,i am child ,i will
leave\n");

    } else {
        printf("parent pid is %d\n",
getpid());

        waitpid(childPid,NULL,0);

        printf("ok,i am father ,i will
leave\n");
        return 0;
    }
```

```
}
```

34

> WELCOME! This challenge makes the following asks of you:
>
> - the challenge checks for a specific parent process : binary
> - the challenge will check that output is redirected to a
>   specific file path : /tmp/vwhunj

很简单了

exp

```
#define _GNU_SOURCE
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>
```

```
void pwncollege() {
    //注意最后的NULL
    char* const argv[]=
{"/challenge/embryoio_level34",NULL};
    const char*
file="/challenge/embryoio_level34";
    //char *const envp[]=
{"bljdlc":"ynugcmanvu"};
    execvp(file,argv);
}


int main(int argc,char** argv[],char* env[])
{
    int childPid;

    int old_fd=open("/tmp/vwhunj",O_WRONLY);
    dup2(old_fd,1);

    if (fork() == 0) {
        childPid = getpid();
        printf("child pid is %d\n",
childPid);

        pwncollege();

        printf("ok,i am child ,i will
leave\n");
```

```
    } else {
        printf("parent pid is %d\n",
getpid());

        waitpid(childPid,NULL,0);

        printf("ok,i am father ,i will
leave\n");
        return 0;
    }
}
```

35

改改exec函数就行了

```
#define _GNU_SOURCE
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>

void pwncollege() {
    //注意最后的NULL
    char* const argv[]=
{"/challenge/embryoio_level35",NULL};
```

```
    const char*
file="/challenge/embryoio_level35";
    char *const envp[]={NULL};
    //int execvpe(const char *file, char
*const argv[],
    //char *const envp[]);
    execvpe(file,argv,envp);
}


int main(int argc,char** argv[],char* env[])
{
    int childPid;

    if (fork() == 0) {
        childPid = getpid();
        printf("child pid is %d\n",
childPid);

        pwncollege();

        printf("ok,i am child ,i will
leave\n");

    } else {
        printf("parent pid is %d\n",
getpid());

        waitpid(childPid,NULL,0);
        printf("ok,i am father ,i will
leave\n");
        return 0;
    }
}
```

36

竖线就可以了

> Linux shell中的竖线（|）——管道符号
>
> 管道符号，是unix一个很强大的功能，符号为一条竖线："|"。
> 用法：command 1 | command 2 他的功能是把第一个命令command 1执行的结果作为command 2的输入传给command 2

```
hacker@embryoio_level36:~$
/challenge/embryoio_level36 | cat
```

37grep

grep实例

> 1、在当前目录中，查找后缀有 file 字样的文件中包含 test 字符串的文件，并打印出该字符串的行。此时，可以使用如下命令：
>
> grep test *file

exp:

```
~$ /challenge/embryoio_level37 | grep
pwn.college
```

38sed

示例：

> ### 数据的搜寻并显示
>
> 搜索 testfile 有 **oo** 关键字的行：
>
> ```
> $ nl testfile | sed -n
> '/oo/p'
>    5   Google
>    7   Runoob
> ```
>
> 如果 root 找到，除了输出所有行，还会输出匹配行。

exp:

```
hacker@embryoio_level38:~$
/challenge/embryoio_level38 |
sed -n '/pwn/p'
pwn.college{wFy-
0ja6lNCDxbrmldk4NneajSS.0FOzwCO5IzW}

#或者
~$ /challenge/embryoio_level38 | sed -ep
```

39-47|rev-stdout

> WELCOME! This challenge makes the following asks of you:
>
> - the challenge checks for a specific parent process : bash
> - the challenge checks for a specific process at the other end of stdout : rev

stdout

40|cat-stdin

> WELCOME! This challenge makes the following asks of you:
>
> - the challenge checks for a specific parent process : bash
> - the challenge checks for a specific process at the other end of stdin : cat
> - the challenge will check for a hardcoded password over stdin : symxccme

没看明白。

exp:

```
hacker@embryoio_level40:~$ cat | /challenge/embryoio_level40
```

41|rev-stdin

> WELCOME! This challenge makes the following asks of you:
>
> - the challenge checks for a specific parent process :

> bash
> - the challenge checks for a specific process at the other end of stdin : rev
> - the challenge will check for a hardcoded password over stdin : xzgyyfgq

没看明白

discord上说

> with rev you need to explicitly tell it that you're done typing
>
> so type the password
>
> then Enter
>
> then Ctrl+D

rev需要Ctlr+D来告诉它，输入结束了。

exp:

```
hacker@embryoio_level41:~$ rev | /challenge/embryoio_level41
```

42stdin: left; stdout: right

```bash
#!/bin/bash
/challenge/embryoio_level42 | cat
```

43grep

exp

```bash
#!/bin/bash
/challenge/embryoio_level43 | grep
pwn.college
```

44sed

```bash
#!/bin/bash
/challenge/embryoio_level44 | sed '/pwn/p'
```

45rev

```
#!/bin/bash
/challenge/embryoio_level45 | rev
```

46cat

```
#!/bin/bash
cat | /challenge/embryoio_level46
```

然后输入密码

47rev

```
#!/bin/bash
rev | /challenge/embryoio_level47
```

要注意，我们的输入，是经过rev的。也就是说，如果要求的密码是123，那么我们输入的就是321.

48

看了看视频，知道怎么做了。

但是还是有很多疑惑。

计算机的很多东西，知道最好的方法即可，研究下去，会是无穷无尽的。

实验

我的实验：

两个程序：

```
//cli.c
#include<stdio.h>

int main()
{
    printf("i love you\n");
    /*
    注意字符最好不要有空格
    有空格的话，实验里面ser接收到的只有一个字母
i，因为
    遇到了空格
    */
    //printf("ok\n");
    //printf("---------------\n");
    return 0;
}

//ser.c
#include<stdio.h>
#include<stdlib.h>
int main()
{
    char *p;
    p= (char *)malloc(100);
    scanf("%s",p);
    printf("%s\n",p);
    return 0;

}
```

一般来讲，服务端先开启服务，等待客户端的请求。

所以，应该是先开启服务端，服务端等待输入。

接着开启客户端，客户端输出，将输出重定向到服务端。

服务端接收到输入，执行，然后结束程序。

客户端执行完后，也结束程序。

exp

```
In [3]: from pwn import *
   ...: p1=process(['/usr/bin/cat'])
   ...:
p2=process(['/challenge/embryoio_level48'],stdout=p1.stdin)
   ...: p1.interactive()
```

```
from pwn import *
p1=process(['/usr/bin/cat'])
p2=process(['/challenge/embryoio_level48'],stdout=p1.stdin)
p1.interactive()
```

49-59

```
from pwn import *
p1=process(['/usr/bin/grep','pwn.college'])
p2=process(['/challenge/embryoio_level49'],st
dout=p1.stdin)
p1.interactive()
```

50

```
from pwn import *
p1=process(["/usr/bin/sed",'/pwn/p'])
#这里参数有意思，不用再加''
p2=process(['/challenge/embryoio_level50'],st
dout=p1.stdin)
p1.interactive()
```

51

```
from pwn import *
p1=process(["/usr/bin/rev"])
p2=process(['/challenge/embryoio_level51'],st
dout=p1.stdin)
p1.interactive()
```

52

```
from pwn import *
p1=process(['/challenge/embryoio_level52'])
p2=process(['/usr/bin/cat'],stdout=p1.stdin)
p1.interactive()
```

53

```
from pwn import *
p1=process(['/challenge/embryoio_level53'])
p2=process(['/usr/bin/rev'],stdout=p1.stdin)
p1.interactive()
```

密码不用翻转，毕竟rev是接收输入的

54

python

```
from pwn import *
p1=process(['/usr/bin/cat'])
p2=process(['/challenge/embryoio_level54'],st
dout=p1.stdin)
p1.interactive()
```

55

```
from pwn import *
p1=process(['/usr/bin/grep','pwn.college'])
p2=process(['/challenge/embryoio_level55'],st
dout=p1.stdin)
p1.interactive()
```

56

```
from pwn import *
p1=process(['/usr/bin/sed','/pwn/p'])
p2=process(['/challenge/embryoio_level56'],st
dout=p1.stdin)
p1.interactive()
```

57

> hacker@embryoio_level57:~$ /challenge/embryoio_level57 WELCOME! This challenge makes the following asks of you:
>
> - the challenge checks for a specific parent process : python
> - the challenge checks for a specific process at the other end of stdout : rev

等价于：

```
/challenge/embryoio_level57 | rev
```

```
from pwn import *
p1=process(['/usr/bin/rev'])
p2=process(['/challenge/embryoio_level57'],stdout=p1.stdin)
p1.interactive()
```

58

> hacker@embryoio_level58:~$ /challenge/embryoio_level58 WELCOME! This challenge makes the following asks of you:
>
> - the challenge checks for a specific parent process : python
> - the challenge checks for a specific process at the other end of stdin : cat
> - the challenge will check for a hardcoded password over stdin : fnxcvtvs

exp

```
from pwn import *
p1=process(['/challenge/embryoio_level58'])
p2=process(['/usr/bin/cat'],stdout=p1.stdin)
p1.interactive()
```

填密码

相当于:

```
cat | /challenge/embryoio_level58
```

59

> WELCOME! This challenge makes the following asks of you:
>
> - the challenge checks for a specific parent process : python
> - the challenge checks for a specific process at the other end of stdin : rev
> - the challenge will check for a hardcoded password over stdin : onudwufy

exp

```
from pwn import *
p1=process(['/challenge/embryoio_level59'])
p2=process(['/usr/bin/rev'],stdout=p1.stdin)
p1.interactive()
```

填密码，不用rev

相当于：

```
rev | /challenge/embryoio_level58
```

60-63

分析：相当于：

```
/challenge/embryoio_level60 | cat
```

/challenge/embryoio_level58 的输出不是1，而是cat

cat的输入不是0，而是/challenge/embryoio_level58。

在这里，使用了pipe()

Function: pipe()

原型：

```
int pipe(int fds[2]);

Parameters :
fd[0] will be the fd(file descriptor) for the
read end of pipe.
fd[1] will be the fd for the write end of
pipe.
Returns : 0 on Success.
-1 on error.
```

exp

注意一点啊，就是可执行程序，也就是/challenge/embryoio_level60必须是子进程，而不是孙子进程。

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>

void pwncollege()
{
    int fd[2];
    pipe(fd);
    if (fork() != 0) {
            dup2(fd[1],1);

            //注意最后的NULL
            char *const argv[] =
{"/challenge/embryoio_level60", NULL};
            const char *file =
"/challenge/embryoio_level60";
            execvp(file, argv);

        } else {
            dup2(fd[0],0);
            char *const argv[] =
{"/usr/bin/cat", NULL};
            const char *file =
"/usr/bin/cat";
            execvp(file, argv);
        }
}

int main(int argc, char **argv[], char
*env[]) {
    int childPid;



    if (fork() == 0) {
        childPid=getpid();
        printf("child pid is %d\n",
getpid());
```

```
        pwncollege();

        printf("ok,i am child ,i will
leave\n");

    } else {
        printf("parent pid is %d\n",
getpid());

        waitpid(childPid, NULL, 0);

        printf("ok,i am father ,i will
leave\n");
        return 0;
    }
}
```

61

```
void pwncollege()
{
    int fd[2];
    pipe(fd);
    if (fork() != 0) {
            dup2(fd[1],1);

            //注意最后的NULL

```

```
                char *const argv[] =
{"/challenge/embryoio_level61", NULL};
                const char *file =
"/challenge/embryoio_level61";
                execvp(file, argv);

        } else {
                dup2(fd[0],0);
                char *const argv[] =
{"/usr/bin/grep","pwn.college",NULL};
                const char *file =
"/usr/bin/grep";
                execvp(file, argv);
        }
}
```

62

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>

void pwncollege()
{
    int fd[2];
    pipe(fd);
    if (fork() != 0) {
            dup2(fd[1],1);
            //注意最后的NULL
            char *const argv[] =
{"/challenge/embryoio_level62", NULL};
            const char *file =
"/challenge/embryoio_level62";
            execvp(file, argv);
        } else {
            dup2(fd[0],0);
            char *const argv[] =
{"/usr/bin/sed", "/pwn/p",NULL};
            const char *file =
"/usr/bin/sed";
            execvp(file, argv);
        }
}

int main(int argc, char **argv[], char
*env[]) {
    int childPid;
    if (fork() == 0) {
        childPid=getpid();
        printf("child pid is %d\n",
getpid());
        pwncollege();
        printf("ok,i am child ,i will
leave\n");
```

```
    } else {
        printf("parent pid is %d\n",
getpid());
        waitpid(childPid, NULL, 0);
        printf("ok,i am father ,i will
leave\n");
        return 0;
    }
}
```

63

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>

void pwncollege()
{
    int fd[2];
    pipe(fd);
    if (fork() != 0) {
        dup2(fd[1],1);
        //注意最后的NULL
```

```
            char *const argv[] =
{"/challenge/embryoio_level63", NULL};
            const char *file =
"/challenge/embryoio_level63";
            execvp(file, argv);
        } else {
            dup2(fd[0],0);
            char *const argv[] =
{"/usr/bin/rev",NULL};
            const char *file =
"/usr/bin/rev";
            execvp(file, argv);
        }
}

int main(int argc, char **argv[], char
*env[]) {
    int childPid;
    if (fork() == 0) {
        childPid=getpid();
        printf("child pid is %d\n",
getpid());
        pwncollege();
        printf("ok,i am child ,i will
leave\n");
    } else {
        printf("parent pid is %d\n",
getpid());
        waitpid(childPid, NULL, 0);
        printf("ok,i am father ,i will leave\n");
        return 0;
    }
}
```

64-65

> WELCOME! This challenge makes the following asks of you:
>
> - the challenge checks for a specific parent process : binary
> - the challenge checks for a specific process at the other end of stdin : cat
> - the challenge will check for a hardcoded password over stdin : qkrlkrdf

相当于:

```
cat | /challenge/embryoio_level64
```

```
```

```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>

void pwncollege()
{
    int fd[2];
    pipe(fd);
    if (fork() != 0) {
            dup2(fd[0],0);
            //注意最后的NULL
            char *const argv[] = 
{"/challenge/embryoio_level64", NULL};
            const char *file = 
"/challenge/embryoio_level64";
            execvp(file, argv);
        } else {
            dup2(fd[1],1);
            char *const argv[] = 
{"/usr/bin/cat",NULL};
            const char *file = 
"/usr/bin/cat";
            execvp(file, argv);
        }
}
```

```
int main(int argc, char **argv[], char
*env[]) {
    int childPid;
    if (fork() == 0) {
        childPid=getpid();
        printf("child pid is %d\n",
getpid());
        pwncollege();
        printf("ok,i am child ,i will
leave\n");
    } else {
        printf("parent pid is %d\n",
getpid());
        waitpid(childPid, NULL, 0);
        printf("ok,i am father ,i will
leave\n");
        return 0;
    }
}
```

输密码

65

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
```

```
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>

void pwncollege()
{
    int fd[2];
    pipe(fd);
    if (fork() != 0) {
            dup2(fd[0],0);
            //注意最后的NULL
            char *const argv[] =
{"/challenge/embryoio_level65", NULL};
            const char *file =
"/challenge/embryoio_level65";
            execvp(file, argv);
        } else {
            dup2(fd[1],1);
            char *const argv[] =
{"/usr/bin/rev",NULL};
            const char *file =
"/usr/bin/rev";
            execvp(file, argv);
        }
}

int main(int argc, char **argv[], char
*env[]) {
    int childPid;
    if (fork() == 0) {
        childPid=getpid();
        printf("child pid is %d\n",
getpid());
        pwncollege();
        printf("ok,i am child ,i will
leave\n");
    } else {
        printf("parent pid is %d\n",
getpid());
        waitpid(childPid, NULL, 0);
        printf("ok,i am father ,i will
leave\n");
        return 0;
    }
}
```

需要翻转密码

66-67|Linux: find

这里的话，查找的字符串要加双引号。不加其实也可以。

但必须全部匹配

```
~$ find /challenge/ -name embryoio_level66 -
exec {} \;
```

;\不能省

67

在66基础上多传个参数而已。

```
find /challenge/ -name embryoio_level67 -exec
{} bellklmpls \;
```

68

先写一个shell脚本：

```
#!/bin/bash
./68
```

68.c:

```
  argvs=(char**)malloc(300*sizeof(char*));
```

上面那一行非常有意思。

我不让编译器报警，警告都不要哈哈哈哈哈哈

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
```

```c
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>

int main() {
    //注意最后的NULL
    char a[10]="aaaaaaa";
    //a=(char)malloc(sizeof(char));
    char** argvs;
    argvs=(char**)malloc(300*sizeof(char*));
    for(int i=0;i<300;i++)
    {
    argvs[i]=(char*)malloc(20*sizeof(char));
    argvs[i]=a;
    }
    char pwd[20]="eccwdpppin";
    printf("%p",pwd);
    argvs[236]=pwd;


    printf("%s",argvs[236]);
    //argvs[236]="eccwdpppin";
    //argvs[276]=NULL;
    const char*
file="/challenge/embryoio_level68";
    execvp(file,argvs);
}
```

关于内存非配问题

写了一下午的C语言代码。

二级指针真的是难理解。

反正就是，一个二级指针，指向一个chunk，这chunk的内容，每8个字节就时一个地址。

这个地址又指向一个又一个chunk，在chunk里面能写东西。

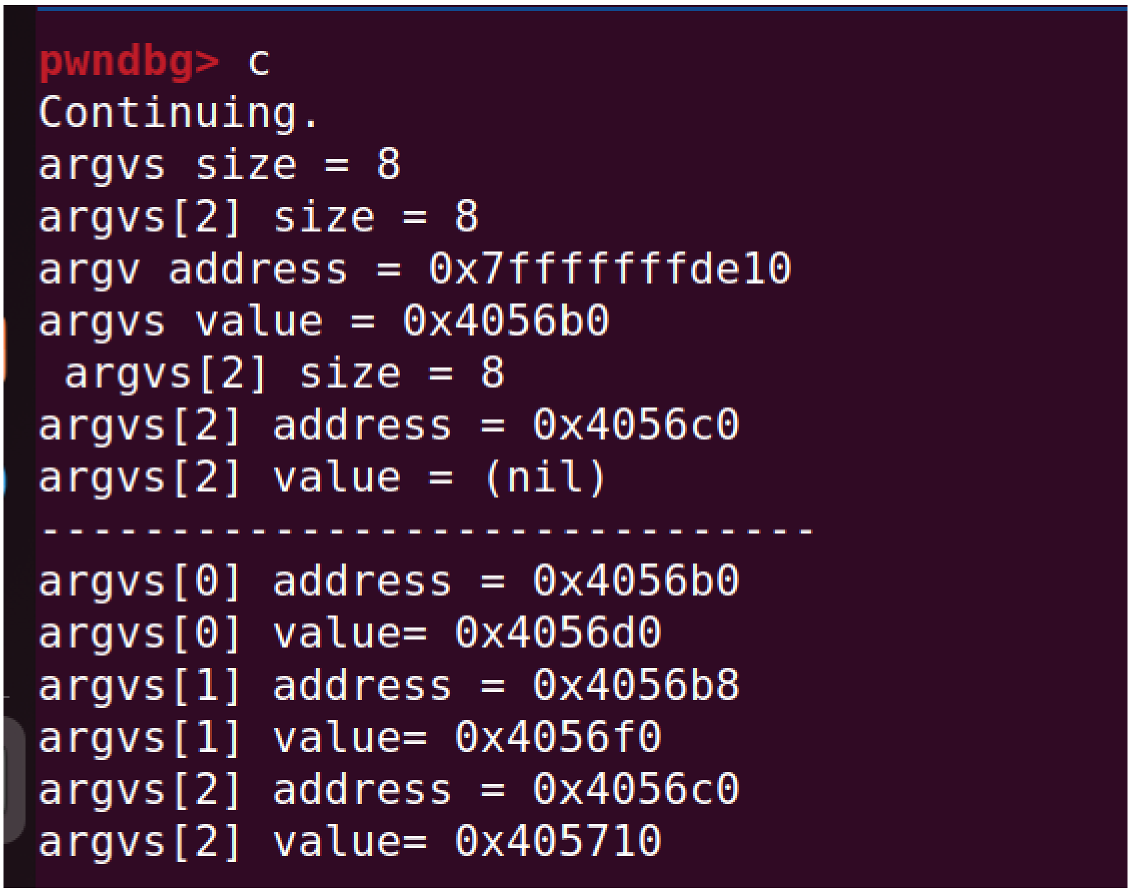
```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <fcntl.h>

int main() {
    //注意最后的NULL
    char a[9]="love";
    //printf
    //a=(char)malloc(sizeof(char));
    char** argvs;
    printf("----------------before malloc
argvs-----------\n");
    printf("argvs size =
%ld\n",sizeof(argvs));
```

```
    printf("argv address = %p\n",&argvs);
    printf("argvs value = %p\n ",argvs);

    printf("argvs[2] size =
%ld\n",sizeof(argvs[2]));
    //无法执行
    //printf("argvs[2] address =
%p\n",&argvs[2]);

    //无法执行
    //printf("argvs[2] value =
%p\n",argvs[2]);

    printf("----------------after malloc
argvs-----------\n");

    argvs=malloc(3*sizeof(char*));
    printf("argv address = %p\n",&argvs);
    printf("argvs value = %p\n ",argvs);

    printf("argvs[2] size =
%ld\n",sizeof(argvs[2]));
    printf("argvs[2] address =
%p\n",&argvs[2]);
    printf("argvs[2] value = %p\n",argvs[2]);
    /*
    argvs[2]=(char*)malloc(8*sizeof(char));
    printf("argvs[2] address =
%p\n",&argvs[2]);
    printf("argvs[2] value = %p\n",argvs[2]);
    //free(argvs);
    //return 0;
    */
    printf("------------then the three
chunks------------------\n");
    printf("\n");
    for(int i=0;i<3;i++)
    {

 printf("begin*************************\n");
        argvs[i]=
(char*)malloc(8*sizeof(char));
        printf("argvs[%d] address =
%p\n",i,&argvs[i]);
        printf("argvs[%d] value= %p\n",i,
argvs[i]);
        printf("argvs[%d] value= %s\n",i,
argvs[i]);
        //执行这一步之后，要注意，每个指针别修改
为a的地址。
        //chunk里面没有内容。
        argvs[i]=a;
        printf("---after---\n");
        printf("argvs[%d] address =
%p\n",i,&argvs[i]);
        printf("argvs[%d] value= %p\n",i,
argvs[i]);
        printf("argvs[%d] value= %s\n",i,
argvs[i]);
```

```
 printf("*************************end\n");

        //printf("after p=a %p\n",&argvs[i]);
    }

    /*
    char pwd[20]="eccwdpppin";
    printf("%p",pwd);
    argvs[236]="";


    printf("%s",argvs[236]);
    //argvs[236]="eccwdpppin";
    //argvs[276]=NULL;
    const char*
file="/challenge/embryoio_level68";
    execvp(file,argvs);
    */
}
```

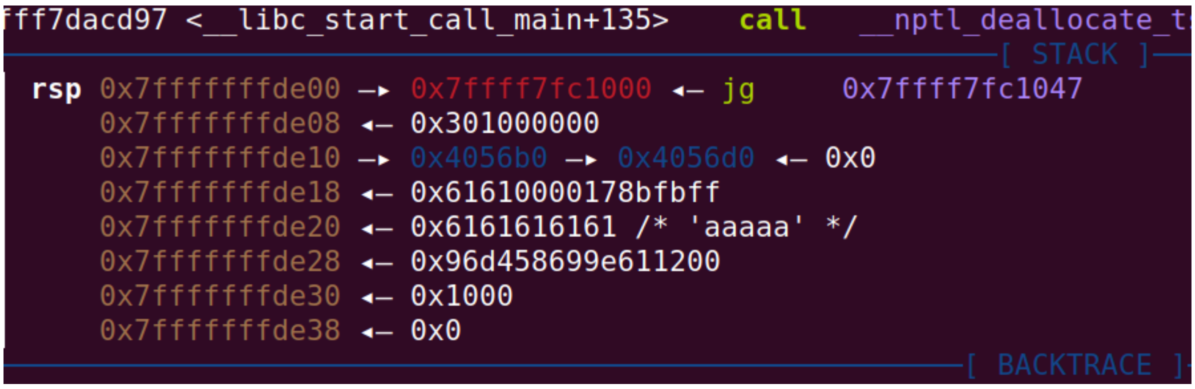
第一次实验:

1



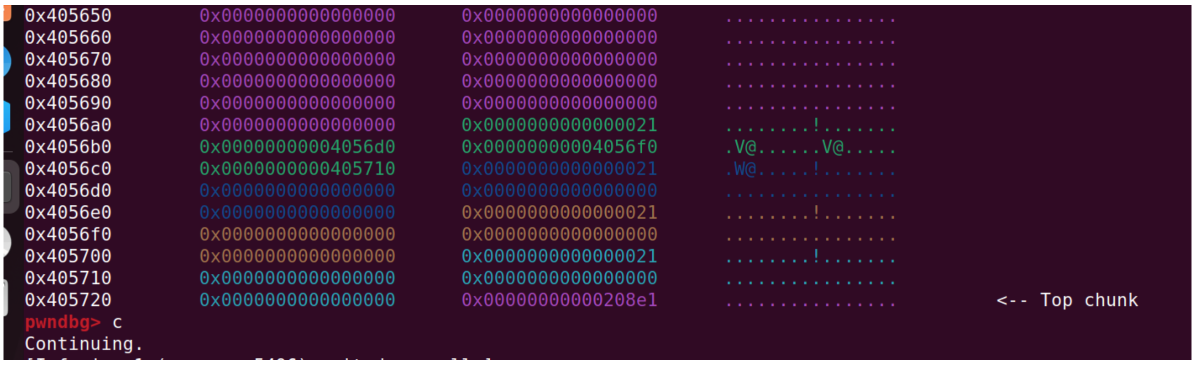
```
pwndbg> c
Continuing.
argvs size = 8
argvs[2] size = 8
argv address = 0x7fffffffde10
argvs value = 0x4056b0
 argvs[2] size = 8
argvs[2] address = 0x4056c0
argvs[2] value = (nil)
------------------------------
argvs[0] address = 0x4056b0
argvs[0] value= 0x4056d0
argvs[1] address = 0x4056b8
argvs[1] value= 0x4056f0
argvs[2] address = 0x4056c0
argvs[2] value= 0x405710
```

2



```
fff7dacd97 <__libc_start_call_main+135>    call   __nptl_deallocate_t
───────────────────────────────────────────────────────[ STACK ]──
 rsp 0x7fffffffde00 —▸ 0x7ffff7fc1000 ◂— jg     0x7ffff7fc1047
     0x7fffffffde08 ◂— 0x301000000
     0x7fffffffde10 —▸ 0x4056b0 —▸ 0x4056d0 ◂— 0x0
     0x7fffffffde18 ◂— 0x6161000178bfbff
     0x7fffffffde20 ◂— 0x6161616161 /* 'aaaaa' */
     0x7fffffffde28 ◂— 0x96d458699e611200
     0x7fffffffde30 ◂— 0x1000
     0x7fffffffde38 ◂— 0x0
───────────────────────────────────────────────────[ BACKTRACE ]─
```

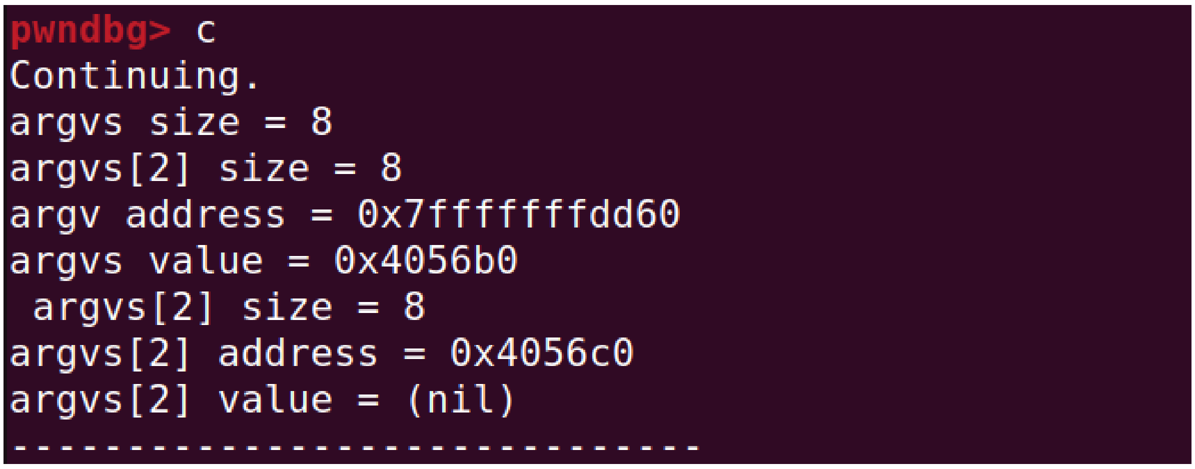
3



```
0x405650    0x0000000000000000    0x0000000000000000    ................
0x405660    0x0000000000000000    0x0000000000000000    ................
0x405670    0x0000000000000000    0x0000000000000000    ................
0x405680    0x0000000000000000    0x0000000000000000    ................
0x405690    0x0000000000000000    0x0000000000000000    ................
0x4056a0    0x0000000000000000    0x0000000000000021    ........!.......
0x4056b0    0x00000000004056d0    0x00000000004056f0    .V@......V@.....
0x4056c0    0x0000000000405710    0x0000000000000021    .W@.....!.......
0x4056d0    0x0000000000000000    0x0000000000000000    ................
0x4056e0    0x0000000000000000    0x0000000000000021    ........!.......
0x4056f0    0x0000000000000000    0x0000000000000000    ................
0x405700    0x0000000000000000    0x0000000000000021    ........!.......
0x405710    0x0000000000000000    0x0000000000000000    ................
0x405720    0x0000000000000000    0x00000000000208e1    ................         <-- Top chunk
pwndbg> c
Continuing.
```

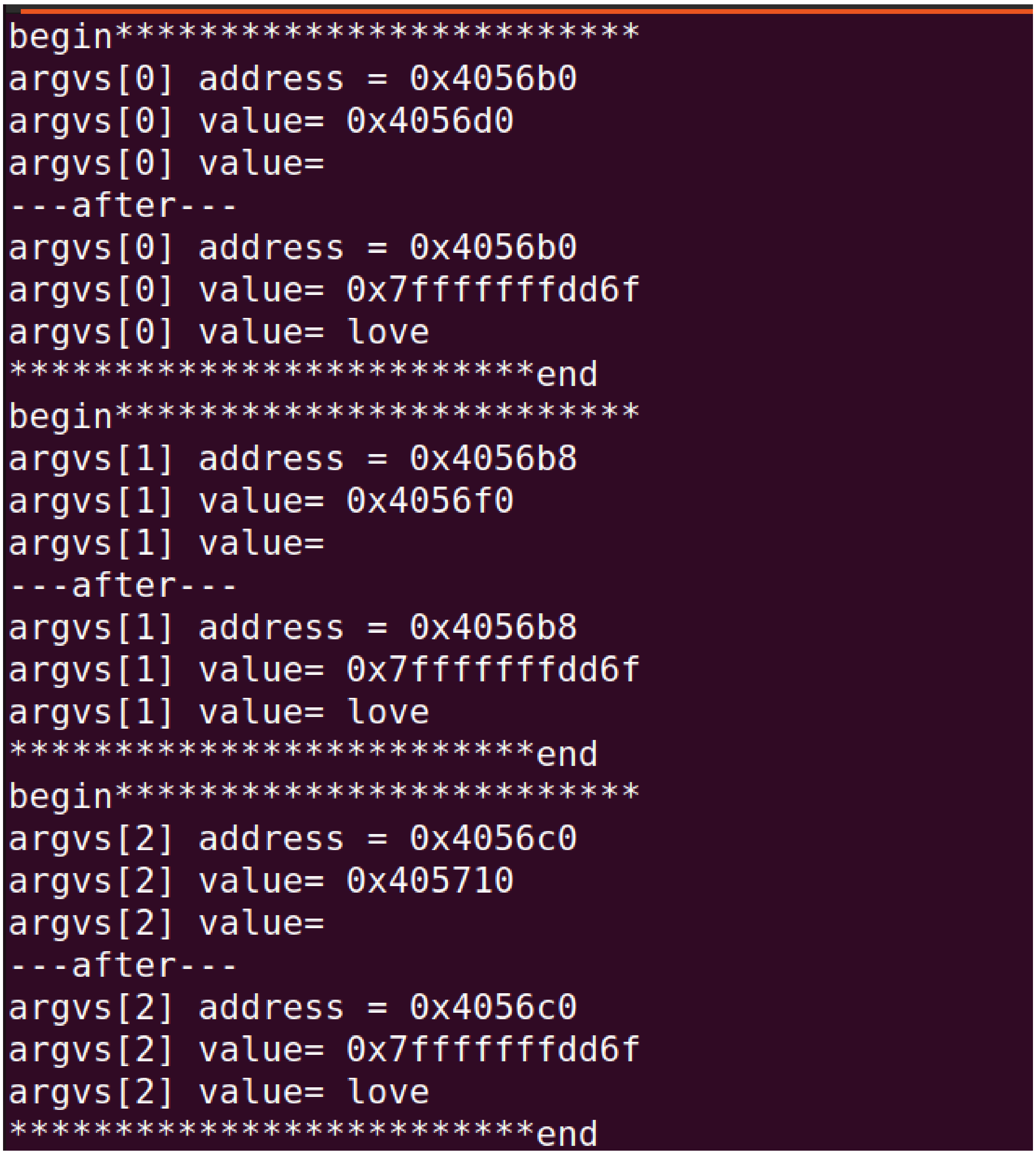
第二次实验：

1



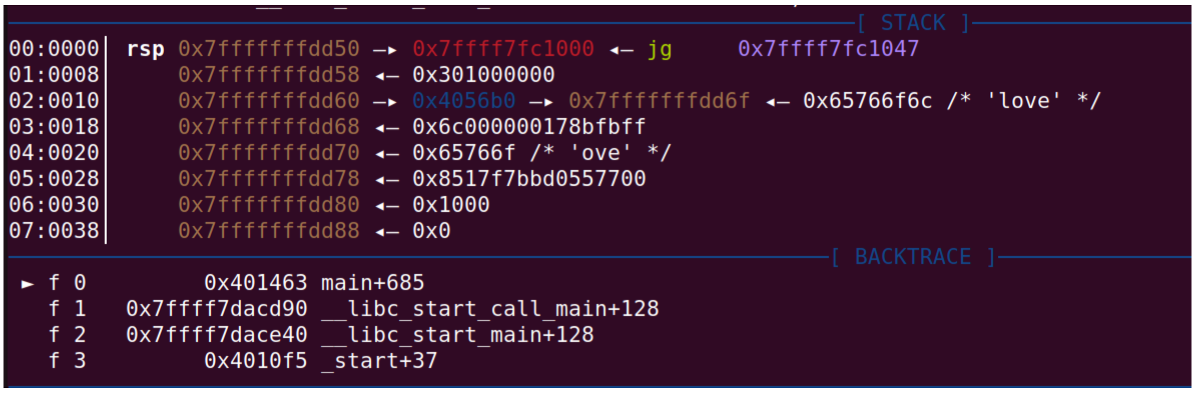
```
pwndbg> c
Continuing.
argvs size = 8
argvs[2] size = 8
argv address = 0x7fffffffdd60
argvs value = 0x4056b0
 argvs[2] size = 8
argvs[2] address = 0x4056c0
argvs[2] value = (nil)
------------------------------
```

2



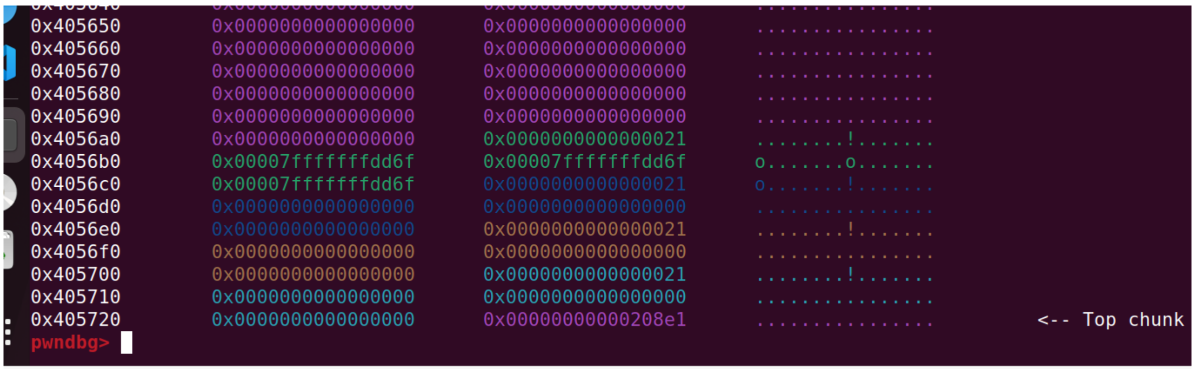
```
begin*************************
argvs[0] address = 0x4056b0
argvs[0] value= 0x4056d0
argvs[0] value=
---after---
argvs[0] address = 0x4056b0
argvs[0] value= 0x7fffffffdd6f
argvs[0] value= love
*************************end
begin*************************
argvs[1] address = 0x4056b8
argvs[1] value= 0x4056f0
argvs[1] value=
---after---
argvs[1] address = 0x4056b8
argvs[1] value= 0x7fffffffdd6f
argvs[1] value= love
*************************end
begin*************************
argvs[2] address = 0x4056c0
argvs[2] value= 0x405710
argvs[2] value=
---after---
argvs[2] address = 0x4056c0
argvs[2] value= 0x7fffffffdd6f
argvs[2] value= love
*************************end
```

3



```
──────────────────────────────────────────────────────────[ STACK ]──────────────
00:0000│  rsp 0x7fffffffdd50 —▸ 0x7ffff7fc1000 ◂— jg     0x7ffff7fc1047
01:0008│      0x7fffffffdd58 ◂— 0x301000000
02:0010│      0x7fffffffdd60 —▸ 0x4056b0 —▸ 0x7fffffffdd6f ◂— 0x65766f6c /* 'love' */
03:0018│      0x7fffffffdd68 ◂— 0x6c000000178bfbff
04:0020│      0x7fffffffdd70 ◂— 0x65766f /* 'ove' */
05:0028│      0x7fffffffdd78 ◂— 0x8517f7bbd0557700
06:0030│      0x7fffffffdd80 ◂— 0x1000
07:0038│      0x7fffffffdd88 ◂— 0x0
────────────────────────────────────────────────────────[ BACKTRACE ]─────────────
 ► f 0         0x401463 main+685
   f 1   0x7ffff7dacd90 __libc_start_call_main+128
   f 2   0x7ffff7dace40 __libc_start_main+128
   f 3         0x4010f5 _start+37
```

4



```
0x405650        0x0000000000000000      0x0000000000000000      ................
0x405660        0x0000000000000000      0x0000000000000000      ................
0x405670        0x0000000000000000      0x0000000000000000      ................
0x405680        0x0000000000000000      0x0000000000000000      ................
0x405690        0x0000000000000000      0x0000000000000000      ................
0x4056a0        0x0000000000000000      0x0000000000000021      ........!.......
0x4056b0        0x00007fffffffdd6f      0x00007fffffffdd6f      o.......o.......
0x4056c0        0x00007fffffffdd6f      0x0000000000000021      o.......!.......
0x4056d0        0x0000000000000000      0x0000000000000000      ................
0x4056e0        0x0000000000000000      0x0000000000000021      ........!.......
0x4056f0        0x0000000000000000      0x0000000000000000      ................
0x405700        0x0000000000000000      0x0000000000000021      ........!.......
0x405710        0x0000000000000000      0x0000000000000000      ................
0x405720        0x0000000000000000      0x00000000000208e1      ................         <-- Top chunk
pwndbg>
```