

操作系统 Linux 系统调用扩充实验报告

小组：202002001021 李天豪 | 202002001037 谭承浩

一、实验目的及分工

扩充 linux0.11 内核 0 版本的系统调用库，并使用测试样例测试所写的系统调用。
实现的系统调用为：

```
1  execve2, getdents, sleep, getcwd
```

小组分工	实现函数	GitHub用户名	commit log 署名
李天豪	sleep, getcwd	yizimi-yuanxin	yizimi
谭承浩	execve2, getdents	TCH-202002001037	TCH

二、实验环境

Windows系统，UbuntuKylin14.10虚拟机，bochs虚拟机，linux0.11内核，VSCode，GitHub

GitHub地址：https://github.com/yizimi-yuanxin/Exper1_ExSysCall

最终版本的分支：dev

三、设计思路

1. execve2

考虑在Linux-0.11已有的系统调用execve的基础上，执行进程时将代码段和数据段都放入内存之中，避免缺页异常的出现。

2. getdents

通过打开文件表filp，找到fd的目录项inode节点号，从而读取几个linux_dirent结构到dirp所指向的缓冲区。

3. sleep

考虑调用alarm来实现。在alarm超时时间时，接收到一个SIGALRM的信号，来判断是否结束，且使用SIG_IGN忽略发出信号时所调用的函数。

4. getcwd

对于getcwd的功能，我们可以从当前目录开始，每次记录他本身的目录项 inode 节点号，然后在该目录中找到 "." 来到达父目录，并在父目录中找到刚刚目录的相同 inode 节点号的目录项，并记录目录名称。依此递归到根节点。

四、实现方法及关键源码

1. execve2

在do_execve函数的基础上进行修改，在进程执行前通过函数immediate()将数据段和代码段都放入内存，从而避免进程执行时出现缺页异常。

```
1  int do_execve2(unsigned long * eip, long tmp, char * filename,
2      char ** argv, char ** envp)
3  {
4      struct m_inode * inode;
5      struct buffer_head * bh;
6      struct exec ex;
7      unsigned long page[MAX_ARG_PAGES];
8      int i, argc, envc;
9      int e_uid, e_gid;
10     int retval;
11     int sh_bang = 0;
12     unsigned long phy_add_brk;
13     unsigned long p=PAGE_SIZE*MAX_ARG_PAGES-4;
14
15     if ((0xffff & eip[1]) != 0x000f)
```

```

16         panic("execve called from supervisor mode");
17     for (i=0 ; i<MAX_ARG_PAGES ; i++)    /* clear page-table */
18         page[i]=0;
19     if (!(inode=namei(filename)))        /* get executables
inode */
20         return -ENOENT;
21     argc = count(argv);
22     envc = count(envp);
23
24 restart_interp:
25     if (!S_ISREG(inode->i_mode)) {    /* must be regular file */
26         retval = -EACCES;
27         goto exec_error2;
28     }
29     i = inode->i_mode;
30     e_uid = (i & S_ISUID) ? inode->i_uid : current->euid;
31     e_gid = (i & S_ISGID) ? inode->i_gid : current->egid;
32     if (current->euid == inode->i_uid)
33         i >>= 6;
34     else if (current->egid == inode->i_gid)
35         i >>= 3;
36     if (!(i & 1) &&
37         !((inode->i_mode & 0111) && suser())) {
38         retval = -ENOEXEC;
39         goto exec_error2;
40     }
41     if (!(bh = bread(inode->i_dev, inode->i_zone[0]))) {
42         retval = -EACCES;
43         goto exec_error2;
44     }
45     ex = *((struct exec *) bh->b_data); /* read exec-header */
46     if ((bh->b_data[0] == '#') && (bh->b_data[1] == '!') &&
(!sh_bang)) {
47         char buf[1023], *cp, *interp, *i_name, *i_arg;
48         unsigned long old_fs;
49
50         strncpy(buf, bh->b_data+2, 1022);
51         brelse(bh);
52         iput(inode);
53         buf[1022] = '\0';
54         if (cp = strchr(buf, '\n')) {

```

```

55         *cp = '\\0';
56         for (cp = buf; (*cp == ' ') || (*cp == '\\t');
cp++);
57     }
58     if (!cp || *cp == '\\0') {
59         retval = -ENOEXEC; /* No interpreter name found */
60         goto exec_error1;
61     }
62     interp = i_name = cp;
63     i_arg = 0;
64     for ( ; *cp && (*cp != ' ') && (*cp != '\\t'); cp++) {
65         if (*cp == '/')
66             i_name = cp+1;
67     }
68     if (*cp) {
69         *cp++ = '\\0';
70         i_arg = cp;
71     }
72     if (sh_bang++ == 0) {
73         p = copy_strings(envc, envp, page, p, 0);
74         p = copy_strings(--argc, argv+1, page, p, 0);
75     }
76     p = copy_strings(1, &filename, page, p, 1);
77     argc++;
78     if (i_arg) {
79         p = copy_strings(1, &i_arg, page, p, 2);
80         argc++;
81     }
82     p = copy_strings(1, &i_name, page, p, 2);
83     argc++;
84     if (!p) {
85         retval = -ENOMEM;
86         goto exec_error1;
87     }
88     old_fs = get_fs();
89     set_fs(get_ds());
90     if (!(inode=namei(interp))) {
91         set_fs(old_fs);
92         retval = -ENOENT;
93         goto exec_error1;
94     }

```

```

95     set_fs(old_fs);
96     goto restart_interp;
97 }
98 brelse(bh);
99 if (N_MAGIC(ex) != ZMAGIC || ex.a_trsize || ex.a_drsize ||
100     ex.a_text+ex.a_data+ex.a_bss>0x3000000 ||
101     inode->i_size <
102     ex.a_text+ex.a_data+ex.a_syms+N_TXTOFF(ex)) {
103     retval = -ENOEXEC;
104     goto exec_error2;
105 }
106 if (N_TXTOFF(ex) != BLOCK_SIZE) {
107     printk("%s: N_TXTOFF != BLOCK_SIZE. See a.out.h.",
108 filename);
109     retval = -ENOEXEC;
110     goto exec_error2;
111 }
112 if (!sh_bang) {
113     p = copy_strings(envc, envp, page, p, 0);
114     p = copy_strings(argc, argv, page, p, 0);
115     if (!p) {
116         retval = -ENOMEM;
117         goto exec_error2;
118     }
119 }
120 if (current->executable)
121     iput(current->executable);
122 current->executable = inode;
123 for (i=0 ; i<32 ; i++)
124     current->sigaction[i].sa_handler = NULL;
125 for (i=0 ; i<NR_OPEN ; i++)
126     if ((current->close_on_exec>>i)&1)
127         sys_close(i);
128 current->close_on_exec = 0;
129 free_page_tables(get_base(current-
130 >ldt[1]),get_limit(0x0f));
131 free_page_tables(get_base(current-
132 >ldt[2]),get_limit(0x17));
133 if (last_task_used_math == current)
134     last_task_used_math = NULL;
135 current->used_math = 0;

```

```

132     p += change_ldt(ex.a_text,page)-MAX_ARG_PAGES*PAGE_SIZE;
133     p = (unsigned long) create_tables((char *)p,argc,envc);
134     current->brk = ex.a_bss +
135         (current->end_data = ex.a_data +
136         (current->end_code = ex.a_text));
137     current->start_stack = p & 0xfffff000;
138     current->euid = e_uid;
139     current->egid = e_gid;
140     i = ex.a_text+ex.a_data;
141     while (i&0xfff)
142         put_fs_byte(0,(char *) (i++));
143     eip[0] = ex.a_entry;
144     eip[3] = p;
145
146     phy_add_brk = 0;
147     while (phy_add_brk < current->brk)
148     {
149         immediate(1, phy_add_brk + current->start_code);
150         phy_add_brk += PAGE_SIZE;
151     }
152
153     return 0;
154 exec_error2:
155     iput(inode);
156 exec_error1:
157     for (i=0 ; i<MAX_ARG_PAGES ; i++)
158         free_page(page[i]);
159     return(retval);
160 }
161 /*-----这只是一条分隔
   线-----*/
162 void immediate(unsigned long error_code,unsigned long address)
163 {
164     int nr[4];
165     unsigned long tmp;
166     unsigned long page;
167     int block,i;
168
169     address &= 0xfffff000;
170     tmp = address - current->start_code;
171     if (!current->executable || tmp >= current->end_data) {

```

```

172         get_empty_page(address);
173         return;
174     }
175     if (share_page(tmp))
176         return;
177     if (!(page = get_free_page()))
178         oom();
179     block = 1 + tmp/BLOCK_SIZE;
180     for (i=0 ; i<4 ; block++,i++)
181         nr[i] = bmap(current->executable,block);
182     bread_page(page,current->executable->i_dev,nr);
183     i = tmp + 4096 - current->end_data;
184     tmp = page + 4096;
185     while (i-- > 0) {
186         tmp--;
187         *(char *)tmp = 0;
188     }
189     if (put_page(page,address))
190         return;
191     free_page(page);
192     oom();
193 }

```

2. getdents

通过打开文件表filp，找到fd的目录项inode节点号，通过bread函数读取内容，用while循环将读取到的几个linux_dirent结构放到dirp所指向的缓冲区。

```

1  int sys_getdents (unsigned int fd, struct linux_dirent *dirp,
    unsigned int count)
2  {
3      struct m_inode *m_ino;
4      struct buffer_head *buff_hd;
5      struct dir_entry *dir;
6      struct linux_dirent usr;
7      int i, j, res;
8      i = 0;
9      res = 0;
10     m_ino = current->filp[fd]->f_inode;

```

```

11     buff_hd = bread(m_ino->i_dev, m_ino->i_zone[0]);
12     dir = (struct dir_entry *)buff_hd->b_data;
13     while (dir[i].inode > 0)
14     {
15         if (res + sizeof(struct linux_dirent) > count)
16             break;
17         usr.d_ino = dir[i].inode;
18         usr.d_off = 0;
19         usr.d_reclen = sizeof(struct linux_dirent);
20         for (j = 0; j < 14; j++)
21         {
22             usr.d_name[j] = dir[i].name[j];
23         }
24         for(j = 0; j < sizeof(struct linux_dirent); j++){
25             put_fs_byte(((char *)&usr)[j], (char *)dirp +
res);
26             res++;
27         }
28         i++;
29     }
30     return res;
31 }

```

3. sleep

我们先调用 signal 函数，发送出 SIGALRM 信号，并传递 SIG_IGN 参数忽略信号处理函数。然后调用 sys_alarm，然后通过调用 sys_pause 实现进程调度，并在 alarm 超时时间时结束即可。

```

1  int sys_sleep(unsigned int seconds) {          // by yizimi
2      sys_signal(SIGALRM, SIG_IGN, NULL);
3      sys_alarm(seconds);
4      sys_pause();
5      return 0;
6  }

```


4. getcwd

current->pwd 记录了指向当前目录 inode 节点的指针，如果我们 current->pwd == current->root，即当前目录为根目录，直接返回 "/"。如果不是根目录，则一直跳到根目录。首先通过 find_father_dir(自定义) 来找到父目录的 inode 的节点号，记录下设备号(i_dev)，调用 iget 函数来访问父目录。再在父目录中调用 find_same_inode 来查找刚刚与目录 inode 节点号相同的目录项，然后将目录项中的 name 整合到目录中。

我们观察 find_entry 函数，find_entry 函数主要实现的是在该inode节点所指向的目录项中查找名字为 name 的目录项并可以返回目录项。我们可以以 find_entry 为原型，把 match 函数更改为相应的判断条件，实现 find_father_dir, find_same_inode，来满足上述操作。

最后，我们需要把目录项通过 put_fs_byte 将内核态中缓存内容拷贝入用户态的 buf 中。

```
1 // a part of kernel/sys.c
2 #define BUF_MAX 4096
3
4 long sys_getcwd(char * buf, size_t size) { // by yizimi
5     // printk("getcwd");
6     char buf_name[BUF_MAX];
7     char *nowbuf;
8     struct dir_entry * de;
9     struct dir_entry * det;
10    struct buffer_head * bh;
11    nowbuf = (char *)malloc(BUF_MAX * sizeof(char));
12    struct m_inode *now_inode = current->pwd;
13    int idev, inid, block;
14    int prev_inode_num = now_inode->i_num;
15    if (now_inode == current->root)
16        strcpy(nowbuf, "/");
17    while (now_inode != current->root) {
18        bh = find_father_dir(&now_inode, &det);
19        idev = now_inode->i_dev;
20        inid = det->inode;
21        now_inode = iget(idev, inid);
22        bh = find_same_inode(&now_inode, &de, prev_inode_num);
23        prev_inode_num = det->inode;
24        strcpy(buf_name, "/");
25        strcat(buf_name, de->name);
```

```

26         strcat(buf_name, nowbuf);
27         strcpy(nowbuf, buf_name);
28     }
29     int chars = size;
30     char *p1 = nowbuf, *p2 = buf;
31     ++chars;
32     while (chars-- > 0)
33         put_fs_byte(*(p1++), p2++);
34     return (long)buf;
35 }

```

```

1 // a part of namei.c
2 struct buffer_head * find_father_dir(struct m_inode ** dir,
   struct dir_entry ** res_dir) {
3     int entries;
4     int block,i;
5     struct buffer_head * bh;
6     struct dir_entry * de;
7     struct super_block * sb;
8     int namelen = 2;
9     const char name[] = "..";
10 #ifdef NO_TRUNCATE
11     if (namelen > NAME_LEN)
12         return NULL;
13 #else
14     if (namelen > NAME_LEN)
15         namelen = NAME_LEN;
16 #endif
17     entries = (*dir)->i_size / (sizeof (struct dir_entry));
18     *res_dir = NULL;
19     if (!namelen)
20         return NULL;
21     if (namelen==2 && get_fs_byte(name)=='.' &&
   get_fs_byte(name+1)=='.') {
22         if ((*dir) == current->root)
23             namelen=1;
24         else if ((*dir)->i_num == ROOT_INO) {
25             sb=get_super((*dir)->i_dev);
26             if (sb->s_imount) {
27                 iput(*dir);
28                 (*dir)=sb->s_imount;

```

```

29         (*dir)->i_count++;
30     }
31 }
32 }
33 if (!(block = (*dir)->i_zone[0]))
34     return NULL;
35 if (!(bh = bread((*dir)->i_dev, block)))
36     return NULL;
37 i = 0;
38 de = (struct dir_entry *) bh->b_data;
39 while (i < entries) {
40     if ((char *)de >= BLOCK_SIZE+bh->b_data) {
41         brelse(bh);
42         bh = NULL;
43         if (!(block = bmap(*dir, i/DIR_ENTRIES_PER_BLOCK))
44             ||
45             !(bh = bread((*dir)->i_dev, block))) {
46             i += DIR_ENTRIES_PER_BLOCK;
47             continue;
48         }
49         de = (struct dir_entry *) bh->b_data;
50     }
51     // The only changed code there...
52     if ((de->name[0] == '.' && de->name[1] == '.' && de->name[2] == '\0')) {
53         *res_dir = de;
54         return bh;
55     }
56     de++;
57     i++;
58 }
59 brelse(bh);
60 return NULL;
61 }
62 struct buffer_head * find_same_inode(struct m_inode ** dir,
63 struct dir_entry ** res_dir, int yizimi) {
64     int entries;
65     int block, i;
66     struct buffer_head * bh;
67     struct dir_entry * de;
68     struct super_block * sb;

```

```

67     int namelen = 7;
68     const char name[] = "yizimi";
69 #ifdef NO_TRUNCATE
70     if (namelen > NAME_LEN)
71         return NULL;
72 #else
73     if (namelen > NAME_LEN)
74         namelen = NAME_LEN;
75 #endif
76     entries = (*dir)->i_size / (sizeof (struct dir_entry));
77     *res_dir = NULL;
78     if (!namelen)
79         return NULL;
80     if (namelen==2 && name[0]=='.' && name[1]=='.') {
81         if ((*dir) == current->root)
82             namelen=1;
83         else if ((*dir)->i_num == ROOT_INO) {
84             sb=get_super((*dir)->i_dev);
85             if (sb->s_imount) {
86                 iput(*dir);
87                 (*dir)=sb->s_imount;
88                 (*dir)->i_count++;
89             }
90         }
91     }
92     if (!(block = (*dir)->i_zone[0]))
93         return NULL;
94     if (!(bh = bread((*dir)->i_dev,block)))
95         return NULL;
96     i = 0;
97     de = (struct dir_entry *) bh->b_data;
98     while (i < entries) {
99         if ((char *)de >= BLOCK_SIZE+bh->b_data) {
100             brelse(bh);
101             bh = NULL;
102             if (!(block = bmap(*dir,i/DIR_ENTRIES_PER_BLOCK))
||
103                 !(bh = bread((*dir)->i_dev,block))) {
104                 i += DIR_ENTRIES_PER_BLOCK;
105                 continue;
106             }

```

```

107         de = (struct dir_entry *) bh->b_data;
108     }
109     if (yizimi == de->inode) { // The only changed code
        there...
110         *res_dir = de;
111         return bh;
112     }
113     de++;
114     i++;
115 }
116 brelse(bh);
117 return NULL;
118 }

```

五、测试过程及画面

以下测试环境已配置好测试文件，文件夹位置为 /usr/root/1

测试环境：bochs + linux0.11

1. execve2

首先在相应位置打补丁（指令打补丁出现问题，故通过补丁文件自行添加补丁代码）

```

[/usr/root/1]# ./execve2
--do_no_page: address=10000000, pid=40
--do_no_page: address=10004000, pid=40
--syscall: sid=87, pid=40
I am test_echo.
--do_no_page: address=13ffdc40, pid=41
--do_no_page: address=1005ac0c, pid=41
[/usr/root/1]#

```

2. getdents

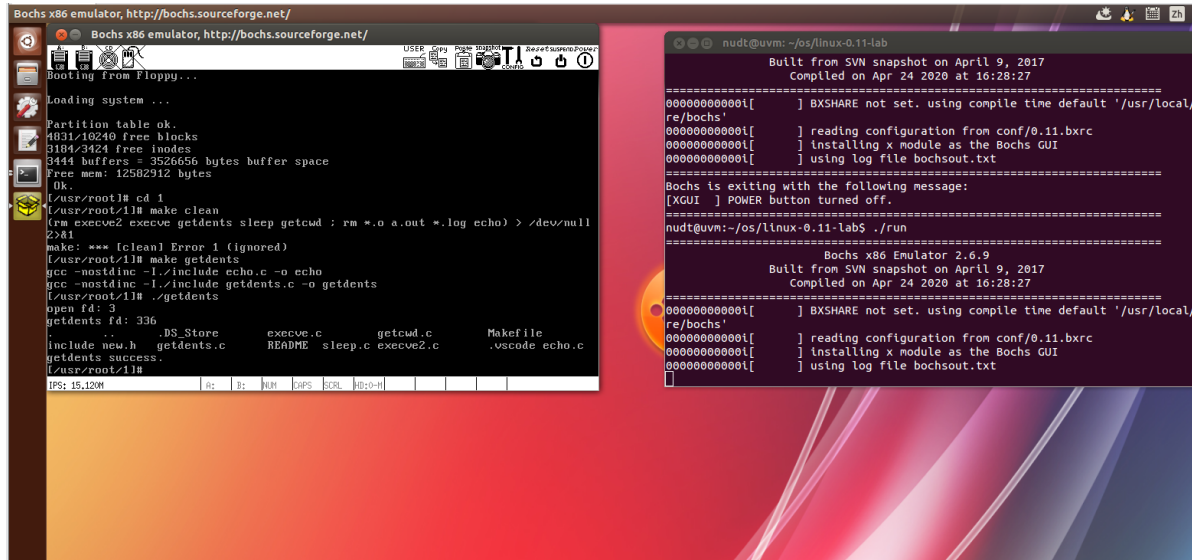
指令（虚拟机开机后）

```

1 cd 1
2 make clean
3 make getdents
4 ./getdents

```

效果：



3. sleep

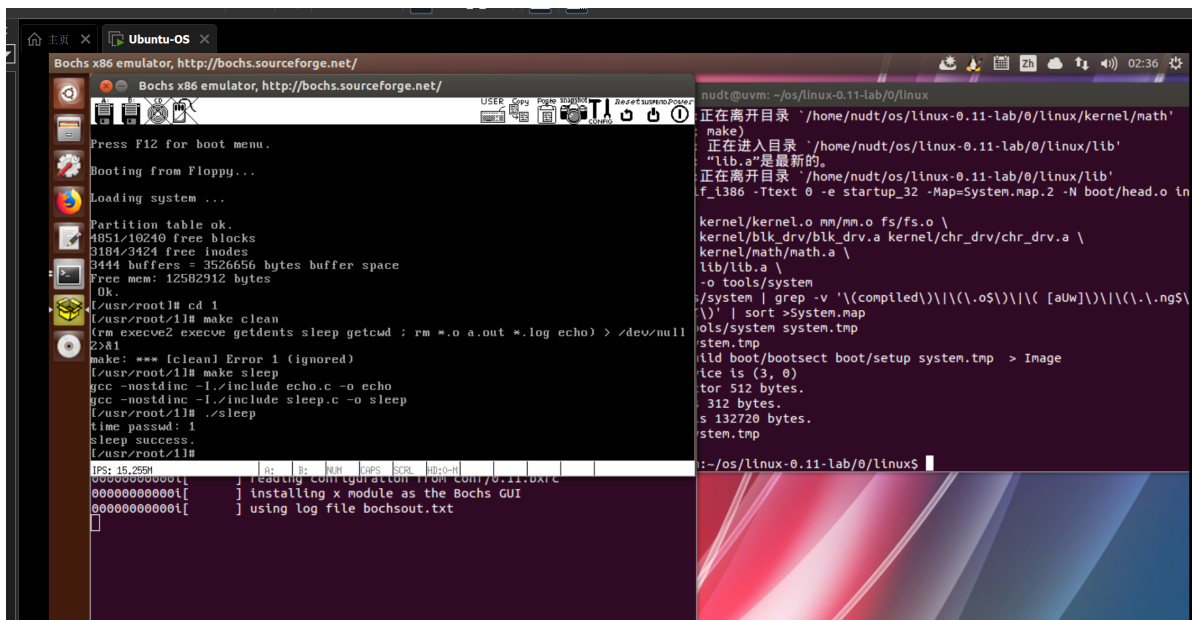
指令（虚拟机开机后）

```

1 cd 1
2 make clean
3 make sleep
4 ./sleep

```

效果：

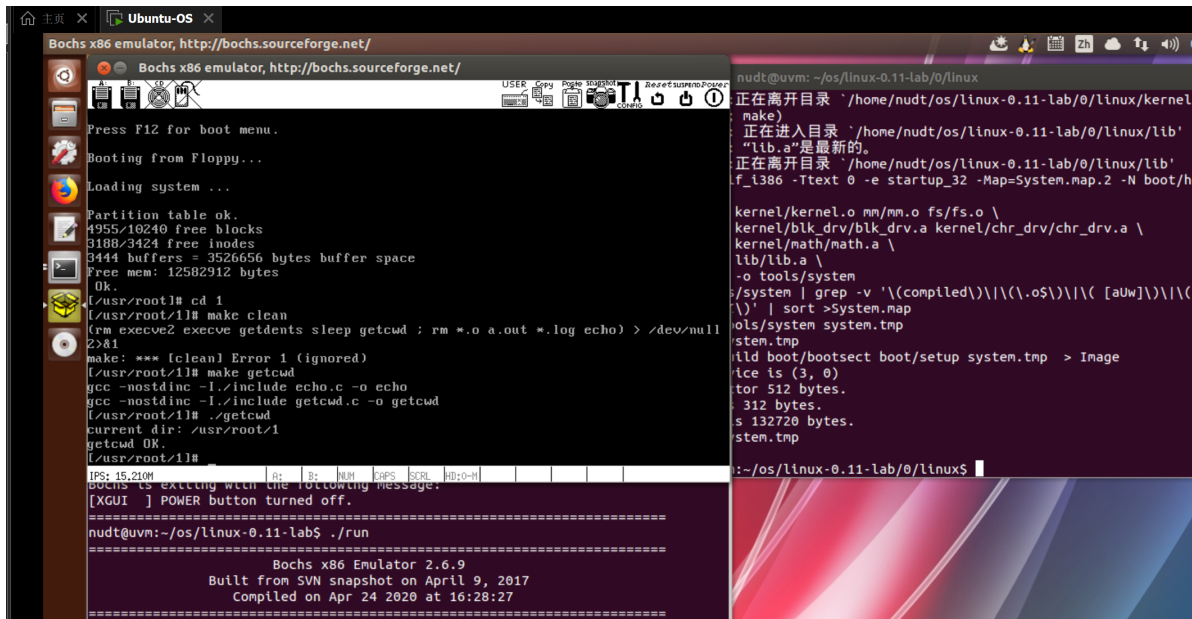


4. getcwd

指令（虚拟机开机后）

```
1 cd 1
2 make clean
3 make getcwd
4 ./getcwd
```

效果：



```
Bochs x86 emulator, http://bochs.sourceforge.net/
Bochs x86 emulator, http://bochs.sourceforge.net/
Press F12 for boot menu.
Booting from Floppy...
Loading system ...
Partition table ok.
4955/10240 free blocks
3188/3424 free inodes
3444 buffers = 3526656 bytes buffer space
Free mem: 12582912 bytes
Ok.
[usr/root/1]# cd 1
[usr/root/1]# make clean
rm execve2 execve getdents sleep getcwd ; rm *.o a.out *.log echo > /dev/null
2>&1
make: *** [clean] Error 1 (ignored)
[usr/root/1]# make getcwd
gcc -nostdinc -I./include echo.c -o echo
gcc -nostdinc -I./include getcwd.c -o getcwd
[usr/root/1]# ./getcwd
current dir: /usr/root/1
getcwd OK.
[usr/root/1]#

Bochs is exiting with the following message.
[XGUIS ] POWER button turned off.
=====
nudit@uvn:~/os/linux-0.11-lab$ ./run
=====
Bochs x86 Emulator 2.6.9
Built from SVN snapshot on April 9, 2017
Compiled on Apr 24 2020 at 16:28:27
=====

nudit@uvn:~/os/linux-0.11-lab/0/linux$
正在离开目录 '/home/nudt/os/linux-0.11-lab/0/linux/kernel'
make)
正在进入目录 '/home/nudt/os/linux-0.11-lab/0/linux/lib'
"lib.a"是最新的。
正在离开目录 '/home/nudt/os/linux-0.11-lab/0/linux/lib'
f_i386 -Ttext 0 -e startup_32 -Map=System.map.2 -N boot/h
kernel/kernel.o mm/mm.o fs/fs.o \
kernel/blk_drv/blk_drv.a kernel/chr_drv/chr_drv.a \
kernel/math/math.a \
lib/lib.a \
-o tools/system
./system | grep -v '\\(compiled\\)\\|\\(\\.o$\\)\\|\\( [aUw]\\)\\|\\(
\\)' | sort >System.map
ols/system system.tmp
stem.tmp
ild boot/bootsect boot/setup system.tmp > Image
ice is (3, 0)
tor 512 bytes.
312 bytes.
s 132720 bytes.
stem.tmp
~/os/linux-0.11-lab/0/linux$
```