

## 实验三 Linux 进程控制 2

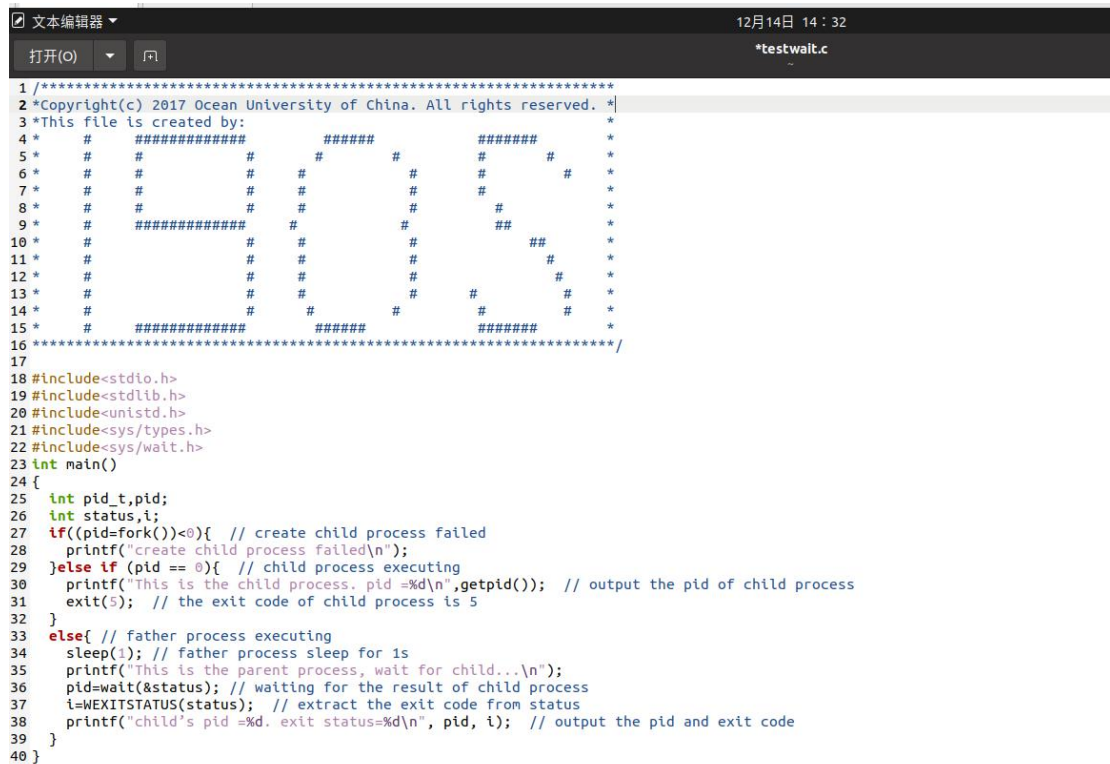
学号: 19020011038 姓名: 岳宇轩 年级: 2019

### 一.使用 wait 系统调用

#### 1.实验要求

- 1) 编写一 C/C++语言程序 (程序名为 testwait.c/testwait.cpp), 使用系统调用 wait()控制进程的阻塞。
- 2) 多次连续反复运行这个程序, 观察屏幕显示结果的顺序, 试简单分析其原因。
- 3) 可以使用实验报告模板中所推荐的代码实现, 但是要求为代码添加注释, 对代码关键逻辑步骤进行解释。在代码头部添加如代码 1 所示式样的头部版权声明。使用星号、井号、等号、破折号等各类符号对版权声明添加边框, 并拼出自己汉字名字的式样。

#### 2.实验结果截图



```
1 /*****  
2 *Copyright(c) 2017 Ocean University of China. All rights reserved. */  
3 *This file is created by:  
4 * #####  
5 * #####  
6 * #####  
7 * #####  
8 * #####  
9 * #####  
10 * #####  
11 * #####  
12 * #####  
13 * #####  
14 * #####  
15 * #####  
16 *****/  
17  
18 #include<stdio.h>  
19 #include<stdlib.h>  
20 #include<unistd.h>  
21 #include<sys/types.h>  
22 #include<sys/wait.h>  
23 int main()  
24 {  
25     int pid_t,pid;  
26     int status,i;  
27     if((pid=fork())<0){ // create child process failed  
28         printf("create child process failed\n");  
29     }else if (pid == 0){ // child process executing  
30         printf("This is the child process. pid =%d\n",getpid()); // output the pid of child process  
31         exit(5); // the exit code of child process is 5  
32     }  
33     else{ // father process executing  
34         sleep(1); // father process sleep for 1s  
35         printf("This is the parent process, wait for child...\n");  
36         pid=wait(&status); // waiting for the result of child process  
37         i=WEXITSTATUS(status); // extract the exit code from status  
38         printf("child's pid =%d. exit status=%d\n", pid, i); // output the pid and exit code  
39     }  
40 }
```

图 1 wait 函数代码

```
yyx@yyx-virtual-machine: ~  
yyx@yyx-virtual-machine:~$ gcc testwait.c -o testwait  
yyx@yyx-virtual-machine:~$ ./testwait  
This is the child process. pid =6067  
This is the parent process, wait for child...  
child's pid =6067. exit status=5  
yyx@yyx-virtual-machine:~$ ./testwait  
This is the child process. pid =6069  
This is the parent process, wait for child...  
child's pid =6069. exit status=5  
yyx@yyx-virtual-machine:~$ ./testwait  
This is the child process. pid =6071  
This is the parent process, wait for child...  
child's pid =6071. exit status=5  
yyx@yyx-virtual-machine:~$ ./testwait  
This is the child process. pid =6073  
This is the parent process, wait for child...  
child's pid =6073. exit status=5  
yyx@yyx-virtual-machine:~$ ./testwait  
This is the child process. pid =6076  
This is the parent process, wait for child...  
child's pid =6076. exit status=5  
yyx@yyx-virtual-machine:~$
```

图 2 wait 函数实验结果

#### 结果分析:

父进程首先创建了一个子进程，然后 sleep 一秒，目的是留给子进程完成工作的时间。

子进程的工作是利用 getpid()函数输出自己的进程号，然后结束自己，并设置退出时的参数为 5.这一过程在父进程的 sleep 1s 内完成。

父进程在 sleep 1s 之后，执行 wait 函数。由于 sleep 一秒的时间之内，子进程可以完成其工作并结束，故 wait()会立即返回子进程结束状态值，再用 WEXITSTATUS()函数获取子进程退出时的状态值，接着将其输出。

#### 尝试情况二:

如果强制增加子进程完成任务所需要的时间，比如在子进程中加入一个 sleep(3)，如图 3 所示。那么可以看到，输出顺序发生了改变，如图 4 所示。

这时，父进程在 sleep 1s 之后，子进程并没有完成工作，故父进程的 wait()函数将父进程阻塞，直到子进程完成，发出信号，父进程才被唤醒。

```

1 /*****
2 *Copyright(c) 2017 Ocean University of China. All rights reserved. *
3 *This file is created by:
4 * # #####
5 * # # # # # #
6 * # # # # # #
7 * # # # # # #
8 * # # # # # #
9 * # ##### # #
10 * # # # # #
11 * # # # # #
12 * # # # # #
13 * # # # # #
14 * # # # # #
15 * # ##### # #
16 *****/
17
18 #include<stdio.h>
19 #include<stdlib.h>
20 #include<unistd.h>
21 #include<sys/types.h>
22 #include<sys/wait.h>
23 int main()
24 {
25     int pid_t,pid;
26     int status,i;
27     if((pid=fork())<0){ // create child process failed
28         printf("create child process failed\n");
29     }else if (pid == 0){ // child process executing
30         asleep(3);
31         printf("This is the child process. pid =%d\n",getpid()); // output the pid of child process
32         exit(5); // the exit code of child process is 5
33     }
34     else{ // father process executing
35         sleep(1); // father process sleep for 1s
36         printf("This is the parent process, wait for child...\n");
37         pid=wait(&status); // waiting for the result of child process
38         i=WEXITSTATUS(status); // extract the exit code from status
39         printf("child's pid =%d. exit status=%d\n", pid, i); // output the pid and exit code
40     }
41 }

```

图 3 wait 函数代码 2

```

yyx@yyx-virtual-machine: ~
yyx@yyx-virtual-machine:~$ gcc testwait.c -o testwait
yyx@yyx-virtual-machine:~$ ./testwait
This is the parent process, wait for child...
This is the child process. pid =6186
child's pid =6186. exit status=5
yyx@yyx-virtual-machine:~$ ./testwait
This is the parent process, wait for child...
This is the child process. pid =6193
child's pid =6193. exit status=5
yyx@yyx-virtual-machine:~$ a

```

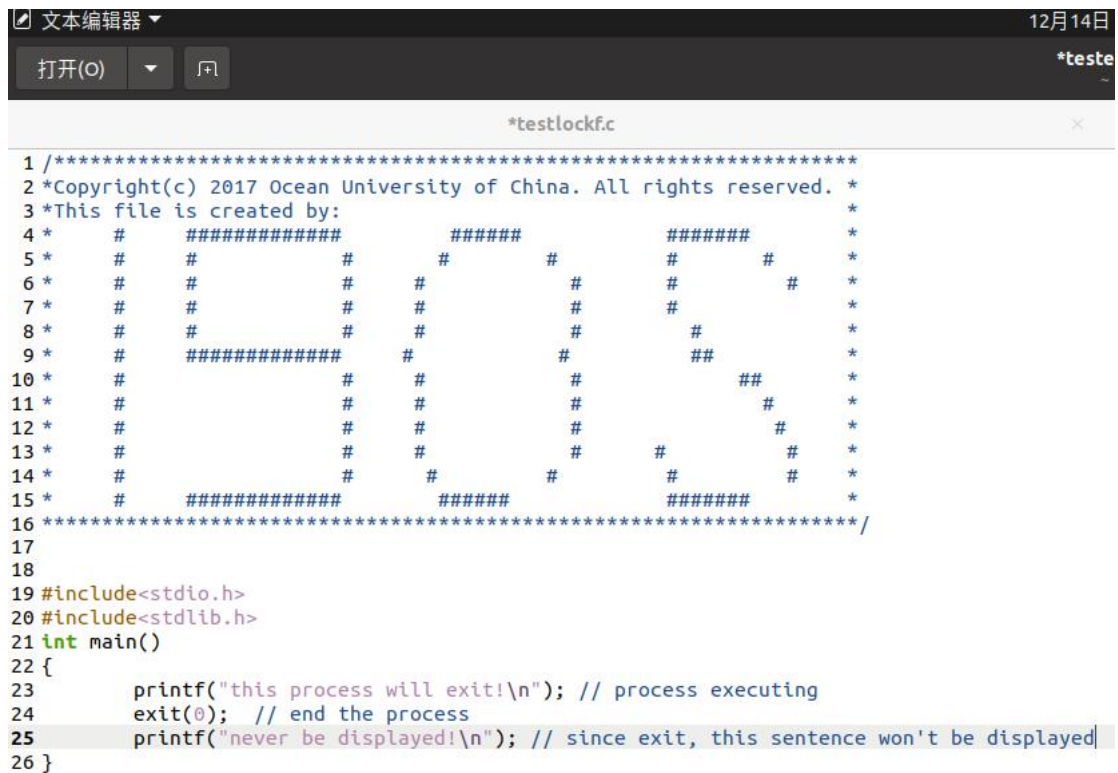
图 4 wait 函数实验结果 2

## 二.使用 exit 系统调用

### 1.实验要求

- 1) 编写一 C/C++ 语言程序（程序名为 testexit.c/testexit.cpp），使用系统调用 exit() 尝试对进程进行终止操作。
- 2) 多次连续反复运行这个程序，观察屏幕显示结果的顺序，试简单分析其原因。
- 3) 可以使用实验报告模板中所推荐的代码实现，但是要求为代码添加注释，对代码关键逻辑步骤进行解释。在代码头部添加如代码 1 所示式样的头部版权声明。使用星号、井号、等号、破折号等各类符号对版权声明添加边框，并拼出自己汉字名字的式样。

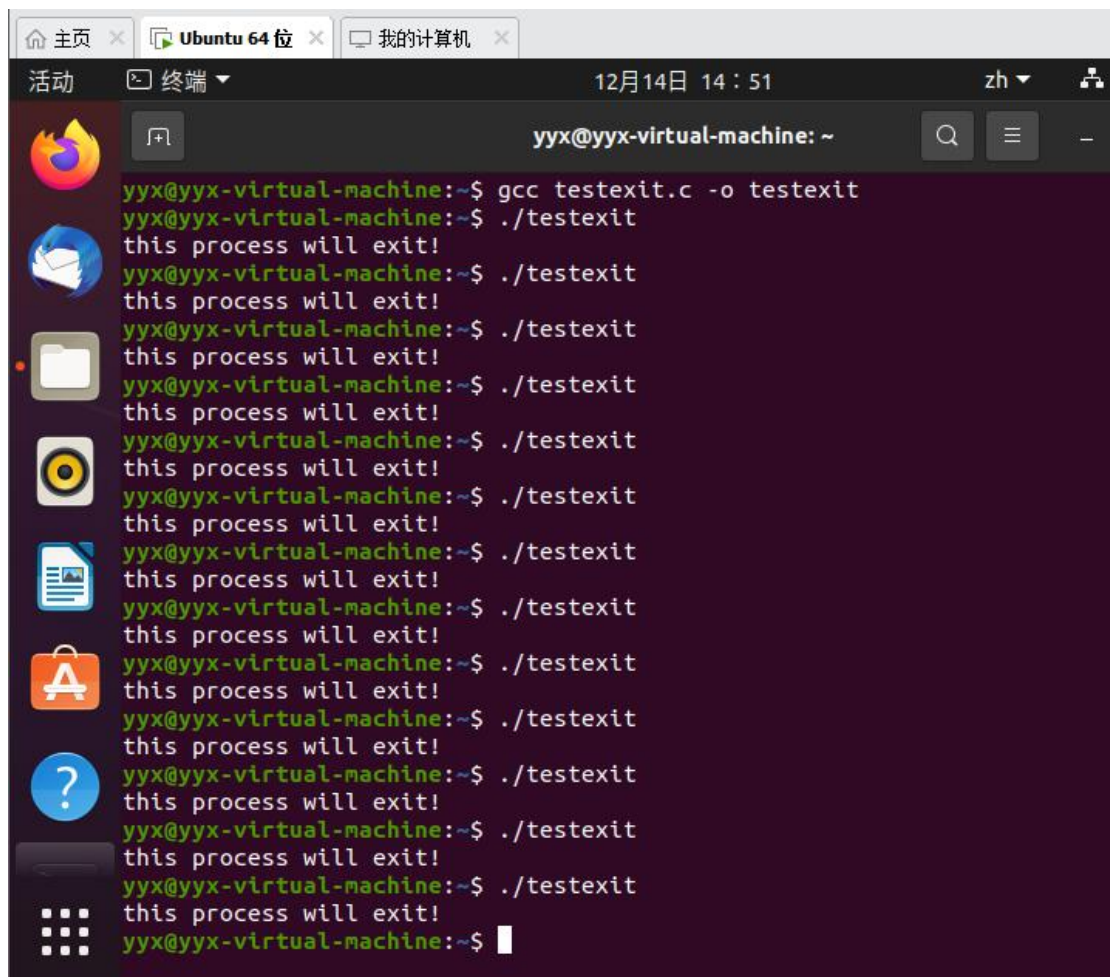
### 2.实验结果截图



```
1 /*****  
2 *Copyright(c) 2017 Ocean University of China. All rights reserved. *  
3 *This file is created by:  
4 * # #####  
5 * # # # # #  
6 * # # # # #  
7 * # # # # #  
8 * # # # # #  
9 * # #####  
10 * # # # # #  
11 * # # # # #  
12 * # # # # #  
13 * # # # # #  
14 * # # # # #  
15 * # #####  
16 *****/  
17  
18  
19 #include<stdio.h>  
20 #include<stdlib.h>  
21 int main()  
22 {  
23     printf("this process will exit!\n"); // process executing  
24     exit(0); // end the process  
25     printf("never be displayed!\n"); // since exit, this sentence won't be displayed  
26 }
```

图 5 exit 函数代码





```
yyx@yyx-virtual-machine:~$ gcc testexit.c -o testexit
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$ ./testexit
this process will exit!
yyx@yyx-virtual-machine:~$
```

图 6 exit 函数代码

### 结果分析：

父进程首先输出了一句“this process will exit! ”。接着结束掉了自己，故之后的“never be displayed!” 将不会被输出。

## 三.使用 lockf 系统调用

### 1.实验要求

- 1) 编写一 C/C++语言程序（程序名为 testlockf.c/testlockf.cpp），使用系统调用 lockf( )尝试对程序的指定区域进行加解锁操作。
- 2) 多次连续反复运行这个程序，观察屏幕显示结果的顺序，试简单分析其原因。
- 3) 可以使用实验报告模板中所推荐的代码实现，但是要求为代码添加注释，对代码关键逻辑步骤进行解释。在代码头部添加如代码 1 所示式样的头部版权声明。使用星号、井号、等号、破折号等各类符号对版权声明添加边框，并拼出自己汉字名字的式样。

### 2.实验结果截图

```
11 *      #      #      #      #      #      *
12 *      #      #      #      #      #      *
13 *      #      #      #      #      #      *
14 *      #      #      #      #      #      *
15 *      #      #####      #####      #####      *
16 *****/
17
18 #include<stdio.h>
19 #include<sys/types.h>
20 #include<unistd.h>
21 int main(void)
22 {
23     int pid1,pid2;
24     int i;
25     lockf(1,1,0); // lock IO resource
26     printf("Parent process:a\n");
27     if((pid1=fork())<0) // child process 1 create failed
28     {
29         printf("child1 fail create\n");
30         return 0;
31     }
32     else if(pid1==0) // child process 1 executing
33     {
34         lockf(1,1,0); // lock IO resource
35         for(i=0;i<5;i++) // output pid for 5 times
36         {
37             printf("This is child1(pid=%d) process:b, number=%d\n",getpid(), i);
38         }
39         lockf(1,0,0); // unlock IO resource
40         return 0;
41     }
42     if((pid2=fork())<0) // child process 2 create failed
43     {
44         printf("child2 fail create\n");
45         return 0;
46     }
47     else if(pid2==0) // child process 2 executing
48     {
49         lockf(1,1,0); // lock IO resource
50         for( i=0;i<5;i++) // output pid for 5 times
51         {
52             printf("This is child2(pid=%d) process:c, number=%d\n",getpid(), i);
53         }
54         lockf(1,0,0); // unlock IO resource
55         return 0;
56     }
57 }
```

图 7 lockf 函数代码

```
yyx@yyx-virtual-machine: ~  
yyx@yyx-virtual-machine:~$ gcc testlockf.c -o testlockf  
yyx@yyx-virtual-machine:~$ ./testlockf  
Parent process:a  
yyx@yyx-virtual-machine:~$ This is child1(pid=7029) process:b, number=0  
This is child1(pid=7029) process:b, number=1  
This is child1(pid=7029) process:b, number=2  
This is child1(pid=7029) process:b, number=3  
This is child1(pid=7029) process:b, number=4  
This is child2(pid=7030) process:c, number=0  
This is child2(pid=7030) process:c, number=1  
This is child2(pid=7030) process:c, number=2  
This is child2(pid=7030) process:c, number=3  
This is child2(pid=7030) process:c, number=4
```

图 8 lockf 函数运行结果

#### 实验结果分析：

lockf(1,1,0)的含义是锁定屏幕输出，lockf(1,0,0)的含义是解锁屏幕输出，第一个参数的含义是 fd 文件；第二个参数，1 表示上锁，0 表示解锁；第三个参数表示文件范围，0 表示全部。

父进程首先锁定了屏幕输出资源，直到父进程结束。故首先输出

Parent process :a

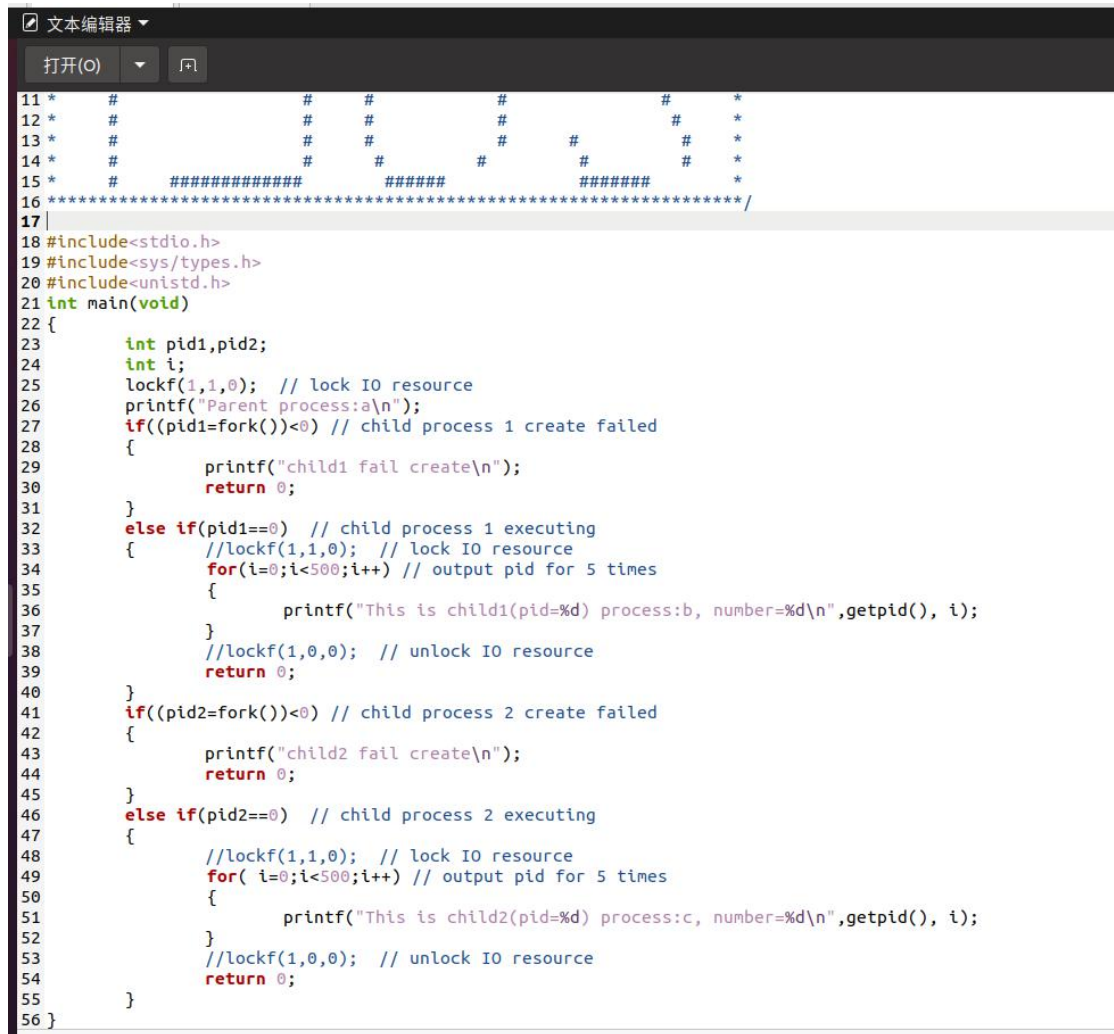
程序中有两个子进程，两个子进程执行相同的操作：1.锁定屏幕输出资源 2.循环 5 次输出自己的 pid 和循环的 number 3.释放屏幕输出资源

考虑进程并发性，两个进程虽然在宏观上是同时进行，但实际上还是有获得处理机的先后顺序的，故输出 b,c 的顺序有可能是交叉的。但因为 b,c 进程在一开始就都使用 lockf 锁定了屏幕输出资源，故即使另一方获得了处理机资源，也无法获得屏幕输出的资源。所以输出 b,c 的顺序是没有交叉的。

#### 尝试改动如下：

去掉子进程中的 lockf，并将循环输出次数均调至 500 次（增加循环次数的原因是尝试使进程在获得一次处理机资源的时间内不能完成所有输出）。

这时，子进程 b,c 应当会出现交叉输出的情况。



```
11 *      #          #          #          #          *
12 *      #          #          #          #          *
13 *      #          #          #          #          *
14 *      #          #          #          #          *
15 *      #          #####          #####          *
16 ***** /
17 |
18 #include<stdio.h>
19 #include<sys/types.h>
20 #include<unistd.h>
21 int main(void)
22 {
23     int pid1,pid2;
24     int i;
25     lockf(1,1,0); // lock IO resource
26     printf("Parent process:a\n");
27     if((pid1=fork())<0) // child process 1 create failed
28     {
29         printf("child1 fail create\n");
30         return 0;
31     }
32     else if(pid1==0) // child process 1 executing
33     {
34         //lockf(1,1,0); // lock IO resource
35         for(i=0;i<500;i++) // output pid for 5 times
36         {
37             printf("This is child1(pid=%d) process:b, number=%d\n",getpid(), i);
38         }
39         //lockf(1,0,0); // unlock IO resource
40         return 0;
41     }
42     if((pid2=fork())<0) // child process 2 create failed
43     {
44         printf("child2 fail create\n");
45         return 0;
46     }
47     else if(pid2==0) // child process 2 executing
48     {
49         //lockf(1,1,0); // lock IO resource
50         for( i=0;i<500;i++) // output pid for 5 times
51         {
52             printf("This is child2(pid=%d) process:c, number=%d\n",getpid(), i);
53         }
54         //lockf(1,0,0); // unlock IO resource
55         return 0;
56     }
57 }
```

图 9 lockf 函数代码 2



```
yyx@yyx-virtual-machine: ~  
This is child1(pid=3096) process:b, number=133  
This is child2(pid=3097) process:c, number=86  
This is child1(pid=3096) process:b, number=134  
This is child2(pid=3097) process:c, number=87  
This is child1(pid=3096) process:b, number=135  
This is child2(pid=3097) process:c, number=88  
This is child1(pid=3096) process:b, number=136  
This is child2(pid=3097) process:c, number=89  
This is child1(pid=3096) process:b, number=137  
This is child2(pid=3097) process:c, number=90  
This is child1(pid=3096) process:b, number=138  
This is child2(pid=3097) process:c, number=91  
This is child1(pid=3096) process:b, number=139  
This is child2(pid=3097) process:c, number=92  
This is child1(pid=3096) process:b, number=140  
This is child2(pid=3097) process:c, number=93  
This is child1(pid=3096) process:b, number=141  
This is child2(pid=3097) process:c, number=94  
This is child1(pid=3096) process:b, number=142  
This is child2(pid=3097) process:c, number=95  
This is child1(pid=3096) process:b, number=143  
This is child2(pid=3097) process:c, number=96  
This is child1(pid=3096) process:b, number=144  
This is child2(pid=3097) process:c, number=97
```

图 10 lockf 函数结果 2