

# 《操作系统》实验报告 (OS-Lab2)

中国人民大学 信息学院 崔冠宇 2018202147

## 一、实验题目： 进程管理.

## 二、实验目的:

- 加深对进程概念的理解,明确进程和程序的区别;
- 进一步认识并发执行的实质.

## 三、实验方法:

在 Linux 兼容环境 (Mac 环境) 下, 按照实验要求, 编写 C 语言程序(详见末尾“程序清单”部分), 主要利用 fork, kill, signal, wait 及 exec 系列函数, 实现相应进程管理和进程间通信功能.

编写完毕后, 使用 gcc 进行编译, 运行可执行文件, 并观察程序输出. 如果和我们的预期相符合, 则该任务完成; 否则仔细分析原因, 查找资料, 修改程序, 直至正确为止.

## 四、程序结构

这四个任务并不复杂, 除了 Linux 系统的默认数据结构(如 pid\_t, pipe 等)之外, 没有用到其他的数据结构, 所以下面主要介绍四个任务我的解决方案, 具体代码请参见末尾的“程序清单”部分.

- 任务一: 这个任务的要求十分简单. 我的思路是: 主函数内, fork 两次, 输出“Parent”; 两个子进程(分别以蓝红两色表示, 下同)中分别输出“Child1”和“Child2”即可, 如下图所示.

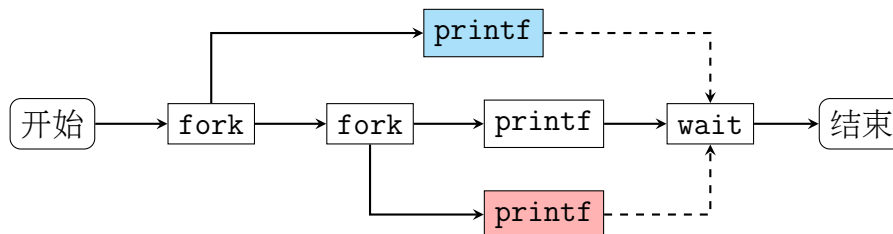


Figure 1: 任务一 主要步骤流程图.

- 任务二: 在主函数内, fork 两个进程. 父进程用 signal 函数注册 SIGINT 信号, 使其收到信号时用 kill 函数, 分别向两个子进程发送 SIGUSR1 和 SIGUSR2 信号, 然后父进程等待子进程全部退出后自己退出; 两个子进程用 signal 函数忽略 SIGINT 信号(避免给父进程发信号时的干扰), 同时注册 SIGUSR1 和 SIGUSR2 信号, 利用 pause 函数等待信号, 使两个进程接收到父进程发来的信号时输出信息然后退出. 如下图所示.

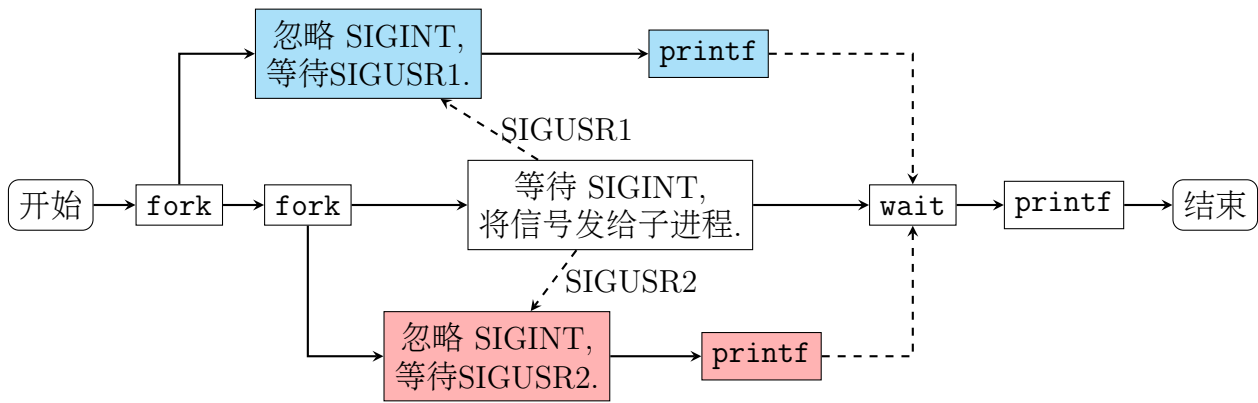


Figure 2: 任务二 主要步骤流程图.

- 任务三: 在主函数内, 先建立起管道, 然后 fork 两个进程. 子进程1向管道写入信息后给父进程发信号 SIGUSR1, 然后等待父进程的信号. 父进程收到子进程1发来的 SIGUSR1 后, 给子进程2发信号 SIGUSR1. 子进程2接到信号后, 向管道写入信息, 然后向父进程发信号 SIGUSR1, 并等待父进程信号. 父进程收到子进程2的信号后, 发信号给两个子进程, 子进程接收到信号后退出, 父进程等待二者退出后退出. 如下图所示.

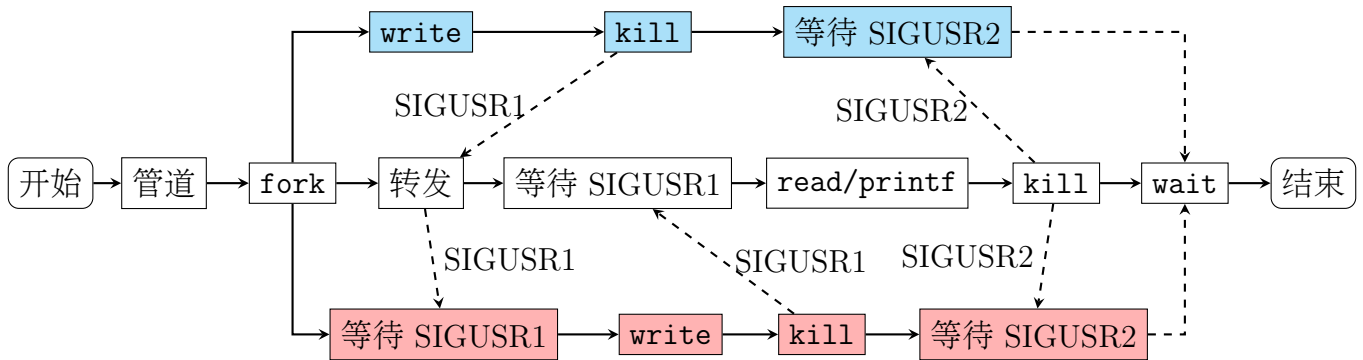


Figure 3: 任务三 主要步骤流程图.

- 任务四: (1) 题目要求写一段多进程并发程序验证加锁的效果. 我打算创建两个进程, 加锁后分别向 stdout 写字符, 最后解锁, 观察效果. 具体流程图如下.

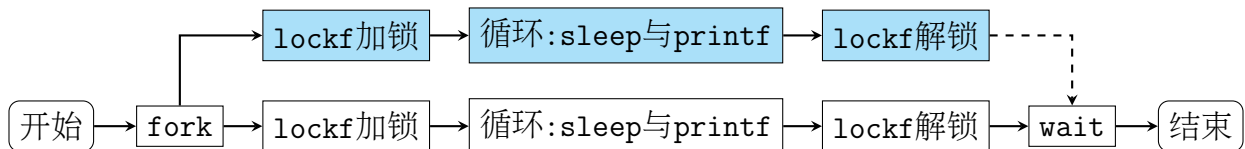


Figure 4: 任务四(1) 主要步骤流程图.

- (2) 只需要在任务三的基础上, 在写操作前给管道加锁, 写完后给管道解锁即可. 此处流程图与任务三变化不大, 故不再列出.

## 五、实验结果

- 任务一的运行结果如下图所示:

```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab2/Code (zsh)
CuiGuanyu@Mac-mini ~/Desktop/Lab2/Code gcc -o task1 task1.c
CuiGuanyu@Mac-mini ~/Desktop/Lab2/Code ./task1
Parent
Child1
Child2
CuiGuanyu@Mac-mini ~/Desktop/Lab2/Code
```

Figure 5: 任务一 运行结果.

程序按照我们的预期显示了“Parent”、“Child1”和“Child2”，可见结果是正确的。

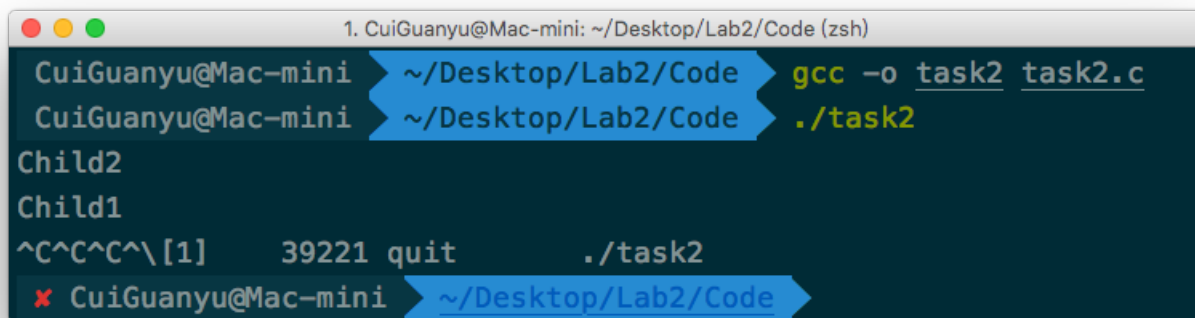
- 任务二的运行结果如下图所示:

```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab2/Code (zsh)
CuiGuanyu@Mac-mini ~/Desktop/Lab2/Code gcc -o task2 task2.c
CuiGuanyu@Mac-mini ~/Desktop/Lab2/Code ./task2
Child1
Child2
^CKilling child1 and child 2...
SIGUSR2 captured!
SIGUSR1 captured!
Child Process 2 is Killed by Parent!
Child Process 1 is Killed by Parent!
Parent Process is killed!
CuiGuanyu@Mac-mini ~/Desktop/Lab2/Code
```

Figure 6: 任务二 运行结果.

根据我们的代码，当用户按下 `ctrl+C` (终端显示 `^C`) 时，`SIGINT` 信号被转给 `sigHandler`，`sigHandler` 函数将 `SIGUSR1` 和 `SIGUSR2` 信号发送给两个子进程，使得两个进程终止，最后父进程终止。由输出可见，除了我增加的一些输出外，程序结果是符合预期的。

如果在主函数的 `signal(SIGINT, sigHandler)` 后面加上 `signal(SIGINT, SIG_IGN)`，则用户再次按下 `ctrl+C` 时，程序由于忽略了中断信号，所以后续不会再有反应，只能手动退出，如下图所示。

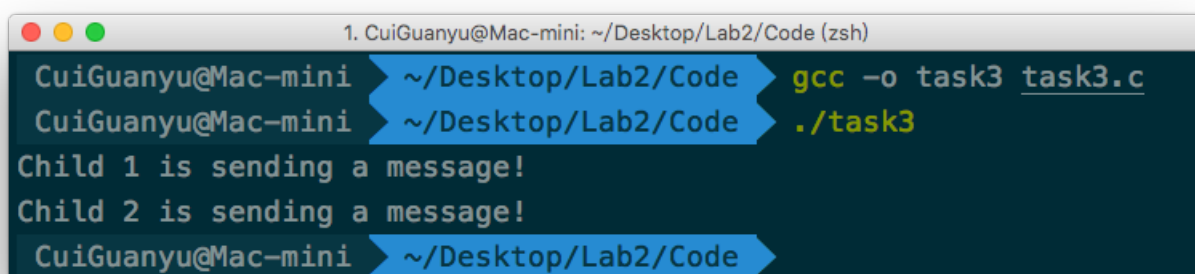


```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab2/Code (zsh)
CuiGuanyu@Mac-mini ~$ gcc -o task2 task2.c
CuiGuanyu@Mac-mini ~$ ./task2
Child2
Child1
^C^C^C^[1] 39221 quit ./task2
CuiGuanyu@Mac-mini ~$
```

Figure 7: 修改 signal 语句的运行结果. ctrl+C 中断无效, 只能 ctrl+\\ 中断.

需要注意的是, 在同一个进程内, 使用 signal 函数对同一个信号注册时, 以较新的为准. 所以更换两句 signal 的顺序, 也会导致结果不同.

- 任务三的运行结果如下图所示:



```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab2/Code (zsh)
CuiGuanyu@Mac-mini ~$ gcc -o task3 task3.c
CuiGuanyu@Mac-mini ~$ ./task3
Child 1 is sending a message!
Child 2 is sending a message!
CuiGuanyu@Mac-mini ~$
```

Figure 8: 任务三 运行结果.

即便在子进程1中有 sleep(1) 语句, 管道中读到的数据仍然是 Child1 在前, 说明我们的程序达到了要求.

- 任务四的运行结果如下图所示:

```
1. CuiGuanyu@Mac-mini: ~/Desktop/Lab2/Code (zsh)
CuiGuanyu@Mac-mini ~/Desktop/Lab2/Code$ gcc task4_1.c -o task4_1
CuiGuanyu@Mac-mini ~/Desktop/Lab2/Code$ ./task4_1
00000.....%
CuiGuanyu@Mac-mini ~/Desktop/Lab2/Code$
```

Figure 9: 任务四(1) lockf实验程序运行结果.

这里我们看到尽管程序中两个循环中都有 sleep 语句,但是由于先进入临界区的进程给 stdout 加了锁,导致另一个进程在解锁前无法访问该资源,所以只有当一个进程输出完并解锁后,另一个进程才能输出.

对于第二问,我们在 write 语句前后增加了 lockf 语句,但是由于我们之前使用了 signal 保证 Child1 先于 Child2,所以输出结果和第三问相比没有变化,这里就不单独截图了.

## 六、问题分析

实验中遇到的问题及解决方法:

1. 在做第一问的时候,我发现虽然代码里写了输出语句,但是运行时两个子进程却没有对应输出,或是仅有一个子进程输出.对照老师给的部分可能用到的函数,我发现自己的程序缺少了 wait 函数,使得父进程还没等到子进程退出便先退出了,改正后程序可以正常运行.
2. 在做第二问的时候,我发现当我按下 ctrl+C 时,子进程也被终止了,什么内容也没有输出.思考检查过后我发现子进程中没用 signal 忽略掉 SIGINT 信号,于是我增加了相应语句,问题得到了解决.
3. 第三问我用了信号来实现子进程的先后顺序,感觉有些绕弯,不知道有没有更好的方法.这个问题还没有解决.

## 七、程序清单

代码也可以在随附的 Code 目录中查看.

### 1. 任务一代码 —— task1.c:

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

```

3 #include <stdlib.h>
4
5 int main(int argc, char *argv[])
6 {
7     printf("Parent\n");
8     // Fork the first child.
9     pid_t child1 = fork();
10    if(child1 == 0)
11    {
12        printf("Child1\n");
13        return 0;
14    }
15    // Fork the second child.
16    pid_t child2 = fork();
17    if(child2 == 0)
18    {
19        printf("Child2\n");
20        return 0;
21    }
22    // Wait all child processes to finish.
23    while(wait(0) != -1);
24    // wait(0); -- Wrong!
25    return 0;
26 }

```

## 2. 任务二代码 —— **task2.c:**

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <signal.h>
5
6 pid_t child1 = (pid_t)0, child2 = (pid_t)0;
7
8 void sigHandler(int sig)
9 {
10    // Signals captured.

```

```

11     if(sig == SIGINT)
12     {
13         printf("Killing child1 and child 2...\n");
14         // Send signals to child processes.
15         kill(child1, SIGUSR1);
16         kill(child2, SIGUSR2);
17     }
18     else if(sig == SIGUSR1)
19     {
20         printf("SIGUSR1 captured!\n");
21     }
22     else if(sig == SIGUSR2)
23     {
24         printf("SIGUSR2 captured!\n");
25     }
26 }
27
28 int main(int argc, char *argv[])
29 {
30     // Child 1.
31     child1 = fork();
32     if(child1 == 0)
33     {
34         //execv("./task2_child1", NULL);
35         printf("Child1\n");
36         // For child, ignore SIGINT
37         signal(SIGINT, SIG_IGN);
38         // Capture SIGUSER1
39         signal(SIGUSR1, sigHandler);
40         // Block
41         pause();
42         printf("Child Process 1 is Killed by Parent!\n");
43         return 0;
44     }
45
46     child2 = fork();

```

```

47     if(child2 == 0)
48     {
49         //execv("./task2_child2", NULL);
50         printf("Child2\n");
51         signal(SIGINT, SIG_IGN);
52         // Capture SIGUSR2
53         signal(SIGUSR2, sigHandler);
54         pause();
55         printf("Child Process 2 is Killed by Parent!\n");
56         return 0;
57     }
58     // Capture SIGINT
59     //signal(SIGINT, SIG_IGN);
60     signal(SIGINT, sigHandler);
61     //signal(SIGINT, SIG_IGN);
62     // Wait child processes.
63     while(wait(0) != -1);
64     printf("Parent Process is killed!\n");
65     return 0;
66 }

```

### 3. 任务三代码 —— **task3.c**:

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <signal.h>
6
7  #define N 256
8
9  void doNothing(int sig) {}
10
11 int main(int argc, char *argv[])
12 {
13     char buffer[N] = {0};
14     // Create a pipe.

```



```

15     int fd[2];
16     pipe(fd);
17
18     pid_t child1 = fork();
19     if(child1 == 0)
20     {
21         char str[] = {"Child 1 is sending a message!\n"};
22         // Test sleep.
23         sleep(1);
24         // Caution: \0
25         write(fd[1], str, strlen(str));
26         // Write done, send parent SIGUSR1.
27         kill(getppid(), SIGUSR1);
28         // Wait for SIGUSR2, then return.
29         signal(SIGUSR2, doNothing);
30         pause();
31         return 0;
32     }
33
34     pid_t child2 = fork();
35     if(child2 == 0)
36     {
37         char str[] = {"Child 2 is sending a message!\n"};
38         // Wait for SIGUSR1, then write data.
39         signal(SIGUSR1, doNothing);
40         pause();
41         write(fd[1], str, strlen(str));
42         // Write done, send parent SIGUSR1.
43         kill(getppid(), SIGUSR1);
44         // Wait for SIGUSR2, then return.
45         signal(SIGUSR2, doNothing);
46         pause();
47         return 0;
48     }
49     // Wait for Child 1's SIGUSR1, then signal Child 2.
50     signal(SIGUSR1, doNothing);

```

```

51     pause();
52     kill(child2, SIGUSR1);
53     // Wait for Child 1's SIGUSR1, then read and print.
54     signal(SIGUSR1, doNothing);
55     pause();
56     read(fd[0], buffer, N);
57     printf("%s", buffer);
58     // Kill Child 1 & 2.
59     kill(child1, SIGUSR2);
60     kill(child2, SIGUSR2);
61     // Wait all child processes.
62     while(wait(0) != -1);
63     return 0;
64 }

```

#### 4. 任务四代码(1) —— **task4\_1.c:**

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <stdlib.h>
4  #include <signal.h>
5
6  int main(int argc, char *argv[])
7  {
8      // Fork a child.
9      pid_t child = fork();
10     if(child == 0)
11     {
12         // Lock stdout, print 5 '.'.
13         lockf(fileno(stdout), F_LOCK, 0);
14         for(int i = 0; i < 5; i++)
15         {
16             sleep(1);
17             putchar('.') ;
18             fflush(stdout);
19         }
20         close(fileno(stdout));

```

```

21         lockf(fileno(stdout), F_ULOCK, 0);
22         return 0;
23     }
24     // Lock stdout, print 5 'o'.
25     lockf(fileno(stdout), F_LOCK, 0);
26     for(int i = 0; i < 5; i++)
27     {
28         sleep(1);
29         putchar('o');
30         fflush(stdout);
31     }
32     // Must close, otherwise the lock won't be removed.
33     close(fileno(stdout));
34     lockf(fileno(stdout), F_ULOCK, 0);
35     wait(0);
36     return 0;
37 }

```

#### 5. 任务四代码(2) —— **task4\_2.c**:

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <signal.h>
6
7 #define N 256
8
9 void doNothing(int sig) {}
10
11 int main(int argc, char *argv[])
12 {
13     char buffer[N] = {0};
14     // Create a pipe.
15     int fd[2];
16     pipe(fd);
17

```

```

18 pid_t child1 = fork();
19 if(child1 == 0)
20 {
21     char str[] = {"Child 1 is sending a message!\n"};
22     // Test sleep.
23     sleep(1);
24     // Caution: \0
25     lockf(fd[1], F_LOCK, 0);
26     write(fd[1], str, strlen(str));
27     lockf(fd[1], F_ULOCK, 0);
28     close(fd[1]);
29     // Write done, send parent SIGUSR1.
30     kill(getppid(), SIGUSR1);
31     // Wait for SIGUSR2, then return.
32     signal(SIGUSR2, doNothing);
33     pause();
34     return 0;
35 }
36
37 pid_t child2 = fork();
38 if(child2 == 0)
39 {
40     char str[] = {"Child 2 is sending a message!\n"};
41     // Wait for SIGUSR1, then write data.
42     signal(SIGUSR1, doNothing);
43     pause();
44     lockf(fd[1], F_LOCK, 0);
45     write(fd[1], str, strlen(str));
46     lockf(fd[1], F_ULOCK, 0);
47     close(fd[1]);
48     // Write done, send parent SIGUSR1.
49     kill(getppid(), SIGUSR1);
50     // Wait for SIGUSR2, then return.
51     signal(SIGUSR2, doNothing);
52     pause();
53     return 0;

```

```
54     }
55     // Wait for Child 1's SIGUSR1, then signal Child 2.
56     signal(SIGUSR1, doNothing);
57     pause();
58     kill(child2, SIGUSR1);
59     // Wait for Child 1's SIGUSR1, then read and print.
60     signal(SIGUSR1, doNothing);
61     pause();
62     read(fd[0], buffer, N);
63     printf("%s", buffer);
64     // Kill Child 1 & 2.
65     kill(child1, SIGUSR2);
66     kill(child2, SIGUSR2);
67     // Wait all child processes.
68     while(wait(0) != -1);
69     return 0;
70 }
```