

# 同济大学计算机系

## 操作系统实验报告



实验内容 UNIX V6++中新进程创建与父子进程同步

学 号 2251745

姓 名 张宇

专 业 计算机科学与技术

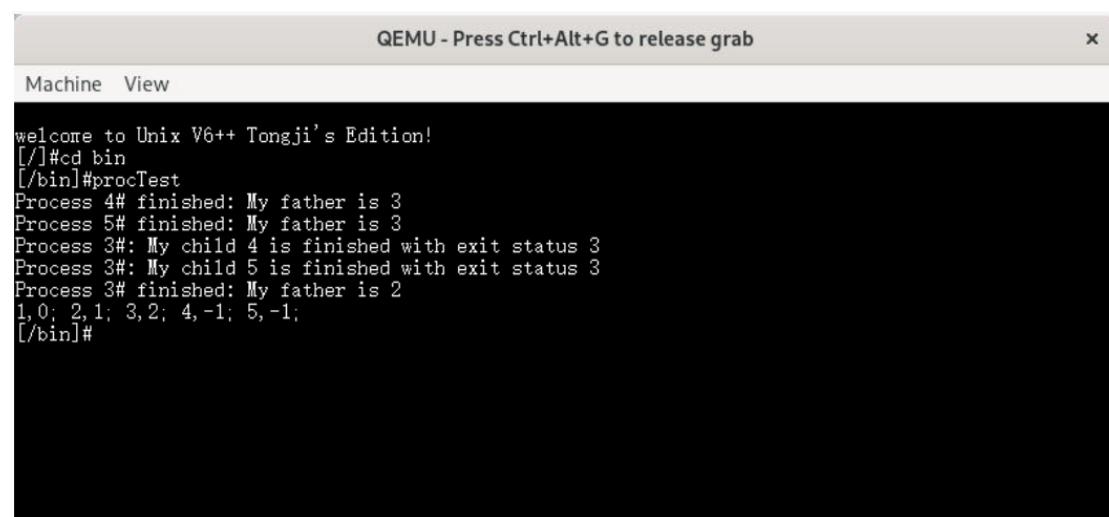
授课老师 方钰

# 一、实验 4.1~实验 4.3

1 具体代码如下：

```
programs > C procTest.c ...
1  #include <stdio.h>
2  #include <sys.h>
3  void main1() {
4      int ws = 2;
5      int i, j, k, pid, ppid;
6
7      if (fork()) {
8          // 2#
9          sleep(2);
10         for (k = 1; k < 6; k++) {
11             printf("%d, %d; ", k, getppid(k));
12         }
13         printf("\n");
14     } else {
15         // 3#
16         if (fork()) {
17             if (fork()) {
18                 // 3#
19                 pid = getpid();
20                 ppid = getppid(pid);
21                 for (k = 0; k < ws; k++) {
22                     i = wait(&j);
23                     printf("Process %d# My child %d is finished with exit status %d\n", pid, i, j);
24                 }
25                 printf("Process %d# finished: My father is %d\n", pid, ppid);
26                 exit(ppid);
27             } else {
28                 // 5#
29                 pid = getpid();
30                 ppid = getppid(pid);
31                 printf("Process %d# finished: My father is %d\n", pid, ppid);
32                 exit(ppid);
33             }
34         } else {
35             // 4#
36             pid = getpid();
37             ppid = getppid(pid);
38             printf("Process %d# finished: My father is %d\n", pid, ppid);
39             exit(ppid);
40         }
41     }
42 }
43 }
```

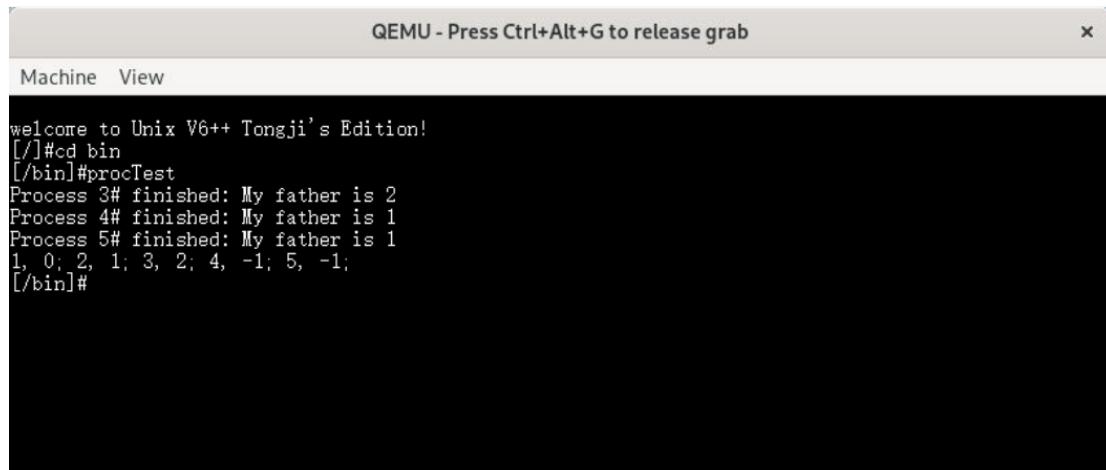
运行结果如下：



The screenshot shows a terminal window titled "QEMU - Press Ctrl+Alt+G to release grab". The window has tabs "Machine" and "View". The terminal output is as follows:

```
welcome to Unix V6++ Tongji's Edition!
[~/]#cd bin
[~/bin]#procTest
Process 4# finished: My father is 3
Process 5# finished: My father is 3
Process 3#: My child 4 is finished with exit status 3
Process 3#: My child 5 is finished with exit status 3
Process 3# finished: My father is 2
1,0; 2,1; 3,2; 4,-1; 5,-1;
[/bin]#
```

2 将 ws 的值修改为 0, 即父进程不执行 wait 后, 运行结果如下:



```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#procTest
Process 3# finished: My father is 2
Process 4# finished: My father is 1
Process 5# finished: My father is 1
1, 0, 2, 1, 3, 2, 4, -1, 5, -1;
[/bin]#
```

3 将 ws 的值修改为 1, 即父进程执行一个 wait 后, 运行结果如下:



```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
welcome to Unix V6++ Tongji's Edition!
[/]#cd bin
[/bin]#procTest
Process 4# finished: My father is 3
Process 5# finished: My father is 3
Process 3#: My child 4 is finished with exit status 3
Process 3# finished: My father is 2
1, 0, 2, 1, 3, 2, 4, -1, 5, 1;
[/bin]#
```

## 二、针对实验 4.3 回答问题

### 1. 进程的调度顺序

3#进程首先创建两个子进程 4#和 5#, 随后执行一次 wait 系统调用, 等待其中一个子进程结束。4#进程在被创建后优先运行, 完成其任务后调用 exit 终止自己, 并唤醒 3#进程, 3#进程随即回收 4#进程的退出码。此时, 5#进程仍然处于活跃状态, 但由于 3#进程只执行了一次 wait 操作, 它并未等待 5#进程结束, 而是直接终止。3#进程终止后, 5#进程被系统重新分配给系统进程 (1#) 作为新的父进程, 直到 5#进程完成任务并由 1#回收。

### 2. 产生这样的调度顺序的原因

这样的调度顺序是由 UNIX V6++的进程管理机制决定的。首先, 4#和 5#作为刚被创建的子进程, 具有相同的初始优先级。在这种情况下, 系统会优先调度进程 ID 较小的进程 (即 4#) 运行, 因此 4#先于 5#执行并结束。3#进程的行为则由代码中 wait 调用的数量决定。由于代码中 3#进程只执行了一次 wait, 它在回收 4#进程后未再等待 5#进程, 而是直接调用 exit 终止自己。这使得 5#进程在没有被 3#回收的情况下被移交给系统进程 1#。

### 3. 5#在最后的打印输出语句时, 为什么显示进程还存在, 且父进程为 1#

这是因为 5#进程在 3#进程终止后尚未结束, 系统自动将 3#的孤立子进程的父进程重置为 1#, 以确保系统中没有孤立的子进程存在。

#### 4. 5#将由谁在何时回收

5#进程将在其调用 exit 终止时，由系统进程 1#回收。

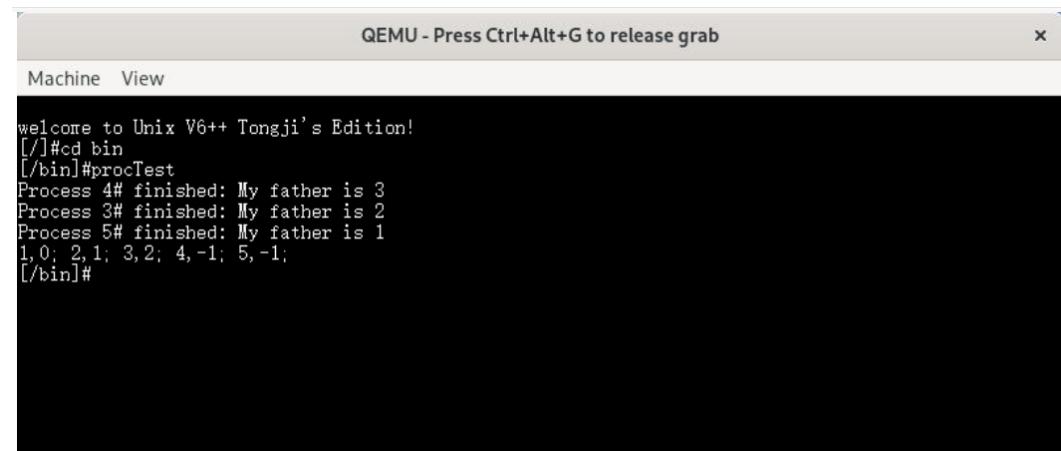
### 三、实验 4.4

#### 1. 对于图 6

修改后的代码如下：

```
programs > C procTest.c > main1()
 1  #include <stdio.h>
 2  #include <sys.h>
 3  void main1() {
 4      int ws = 0;
 5      int i, j, k, pid, ppid;
 6
 7      if (fork()) {
 8          // 2#
 9          sleep(2);
10          for (k = 1; k < 6; k++) {
11              printf("%d,%d ", k, getppid(k));
12          }
13          printf("\n");
14      } else {
15          // 3#
16          if (fork()) {
17              sleep(1);
18              if (fork()) {
19                  // 3#
20                  pid = getpid();
21                  ppid = getppid(pid);
22                  for (k = 0; k < ws; k++) {
23                      i = wait(&j);
24                      printf("Process %d#: My child %d is finished with exit status %d\n", pid, i, j);
25                  }
26                  printf("Process %d# finished: My father is %d\n", pid, ppid);
27                  exit(ppid);
28              } else {
29                  // 5#
30                  pid = getpid();
31                  ppid = getppid(pid);
32                  printf("Process %d# finished: My father is %d\n", pid, ppid);
33                  exit(ppid);
34              }
35          } else {
36              // 4#
37              pid = getpid();
38              ppid = getppid(pid);
39              printf("Process %d# finished: My father is %d\n", pid, ppid);
40              exit(ppid);
41      }
42  }
43 }
44 }
```

运行结果如下：



```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
welcome to Unix V6++ Tongji's Edition!
[]#cd bin
[/bin]#procTest
Process 4# finished: My father is 3
Process 3# finished: My father is 2
Process 5# finished: My father is 1
1,0; 2,1; 3,2; 4,-1; 5,-1;
[/bin]#
```

输出解释：

3#首先执行 fork 创建 4#进程，由于 4#优先级高，立即被调度运行并打印出自己的 ID 和父进程 ID。4#在执行完成后调用 exit 终止，并将其退出码返回给 3#。随后，3#恢复执行，继续调用 fork 创建 5#进程，并打印出自己的 ID 和父进程 ID 后执行 exit 终止，创建的 5#的父进程改为了 1#进程。3#进程终止后，5#进程上台执行，打印出自己的 ID 和父进程 ID 后执行 exit 终止。

原因：在创建 4#进程后，3#进程通过 sleep 等待 1s，所以当前只有一个子进程，其初始优先级为 0（最高优先级），而父进程的优先级因调用 fork 而被重算并降低。因此，4#进程在创建后立即获得调度权，并抢占了 3#的执行。由于此时 5#进程尚未被创建，4#成为唯一活跃的高优先级进程。后续与实验 4.2 相同。

## 2. 对于图 7

修改后的代码如下：

```
programs > C procTest.c > main1()
1  #include <stdio.h>
2  #include <sys.h>
3  void main1() {
4      int ws = 0;
5      int i, j, k, pid, ppid;
6
7      if (fork()) {
8          // 2#
9          sleep(2);
10         for (k = 1; k < 6; k++) {
11             printf("%d,%d ", k, getppid(k));
12         }
13         printf("\n");
14     } else {
15         // 3#
16         if (fork()) {
17             if (fork()) {
18                 // 3#
19                 sleep(1);
20                 pid = getpid();
21                 ppid = getppid(pid);
22                 for (k = 0; k < ws; k++) {
23                     i = wait(&j);
24                     printf("Process %d#: My child %d is finished with exit status %d\n", pid, i, j);
25                 }
26                 printf("Process %d# finished: My father is %d\n", pid, ppid);
27                 exit(ppid);
28             } else {
29                 // 5#
30                 pid = getpid();
31                 ppid = getppid(pid);
32                 printf("Process %d# finished: My father is %d\n", pid, ppid);
33                 exit(ppid);
34             }
35         } else {
36             // 4#
37             pid = getpid();
38             ppid = getppid(pid);
39             printf("Process %d# finished: My father is %d\n", pid, ppid);
40             exit(ppid);
41         }
42     }
43 }
44 }
```

运行结果如下：

```
Machine View  
welcome to Unix V6++ Tongji's Edition!  
[/]#cd bin  
[/bin]#procTest  
Process 4# finished: My father is 3  
Process 5# finished: My father is 3  
Process 3# finished: My father is 2  
1,0, 2,1; 3,2; 4,1, 5,1;  
[/bin]#
```

输出解释：

3#进程执行两个 fork 创建出 4#进程和 5#进程后，4#进程先执行，执行完后 5#进程执行，最后再执行 3#进程。

原因：3#先执行 fork 创建 4#进程和 5#进程后，先执行 sleep 等待 1s，此时有 4#进程和 5#进程，优先级相同时系统优先调用 id 小的进程，因此 4#进程抢占执行权并打印出自己的 ID 和父进程 ID，4#执行完成后调用 exit 终止并返回，此时 3 进程仍然在等待，于是 5#进程获得执行权并打印出自己的 ID 和父进程 ID，5#执行完成后调用 exit 终止并返回，3#进程等待完毕后获得执行权并打印出自己的 ID 和父进程 ID，执行完成后调用 exit 终止，由于 4#进程和 5#进程在终止时，其父进程 3#进程还未终止，因此父进程仍然为 3#进程。