

操作系统 实验报告

实验名称: 实验 1 进程的创建实验

姓名: 刘亚辉

学号: 16340157

实验 1 进程的创建实验

一、实验目的：

1. 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。
2. 认识进程生成的过程，学会使用 fork 生成子进程，并知道如何使子进程完成与父进程不同的工作。

二、实验要求：

1. Linux/Ubuntu 下程序编译和调试实验
2. 编写 makefile 实验

三、实验过程：

1. 运行下面的程序，解释现象。

```
01. #include <linux/types.h>
02. #include <stdio.h>
03. #include <unistd.h>
04. int main(){
05.     int pid1=fork();
06.     printf("**1**\n");
07.
08.     int pid2=fork();
09.     printf("**2**\n");
10.
11.     if(pid1==0){
12.         int pid3=fork();
13.         printf("**3**\n");
14.     }
15.     else
16.         printf("**4**\n");
17.     return 0;
18. }
```

运行，输出结果如下：

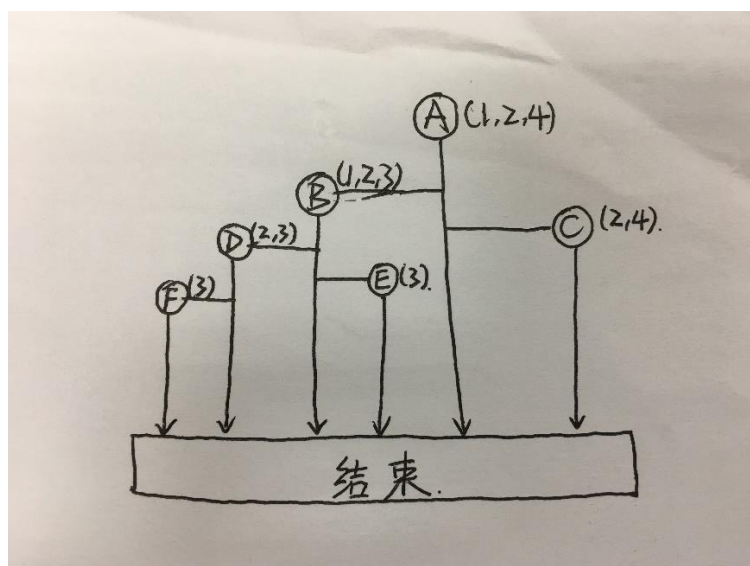
```
01. **1**
02. **1**
03. **2**
04. **2**
05. **3**
06. **3**
07. **3**
08. **2**
09. **4**
10. **2**
11. **4**
12. **3**
```

分析：

一个进程，包括代码、数据和分配给进程的资源。首先解释一下 fork 函数：fork 函数通过系统调用创建一个与原来进程几乎一样的进程，也即在输入相同的情况下，两个进程可以做完全相同的事情。fork 函数是把当前的情况拷贝一份，也就是子进程是从 fork 函数调用之后的那一行开始执行的。另外，fork 函数调用一次，却能够返回两个，并且可以有三种不同的返回值：

- (1) 在父进程中，fork 返回的是创建的子进程的 id；
- (2) 在子进程中，fork 的返回 0；
- (3) 如果出现错误，fork 返回一个负值。

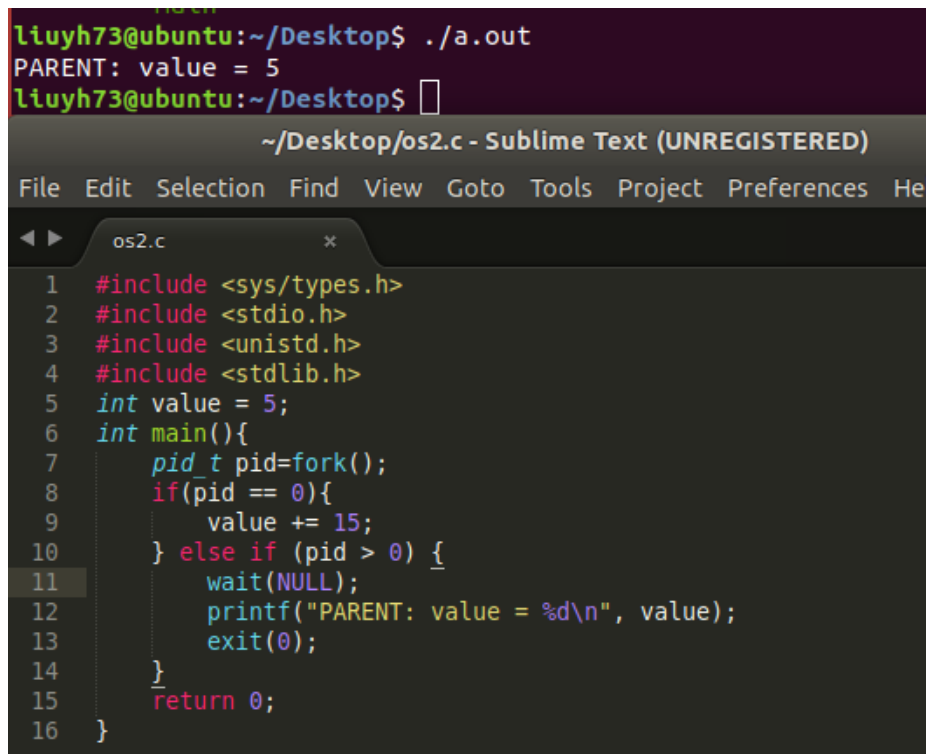
所以可以通过 fork 的返回值来判断当前进程是子进程还是父进程。整个程序运行的时间轴如下：



程序开始运行的时候，只有一个进程 A，在运行到第五行的时候调用 fork 函数创建了一个 A 的子进程 B。先考虑 A 的执行情况：pid1 在进程 A 中不为 0，之后，进程 A 输出了数字 1；然后又调用了 fork 函数，创建了另一个 A 的子进程 C，且 pid2 在进程 A 中也不为 0，之后，进程 A 输出了数字 2；if 条件句 pid1==0 不成立，所以进程 A 输出了 4，至此进程 A 在调用了 return 0 后结束。然后考虑 A 的子进程 B，在创建 B 之后，pid1 在 B 中为 0，然后输出数字 1；调用 fork 函数创建 B 的子进程 D，然后输出 2；if 条件句成立，再次调用 fork 函数创建进程 E，并且输出 3，最终进程 B 结束。然后考虑进程 E，进程 E 再创建之后输出 3，之后结束。再来考虑进程 D，进程 D 创建之后保留了其父进程 B 的 pid1=0，所以输出了 2 之后，条件句判断成立，之后调用 fork 函数创建了进程 F，

并且输出了 3，结束。进程 F 和 E 一样，在输出了 3 之后结束。最后考虑进程 C，进程 C 创建之后保留了其父进程 A 的 pid1 不等于 0，所以输出了 2 之后 if 条件句不成立，所以输出 4，结束。最终整个程序代码运行结束。

2. 题目 3.4



```
liuyh73@ubuntu:~/Desktop$ ./a.out
PARENT: value = 5
liuyh73@ubuntu:~/Desktop$
```

```
~/Desktop/os2.c - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

os2.c
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  int value = 5;
6  int main(){
7      pid_t pid=fork();
8      if(pid == 0){
9          value += 15;
10     } else if (pid > 0) {
11         wait(NULL);
12         printf("PARENT: value = %d\n", value);
13         exit(0);
14     }
15     return 0;
16 }
```

由上图运行结果可知，最终 value 的输出结果为 5；原因：在程序最初运行开始时，value=5，声明为全局变量；然后调用 fork 函数创建了一个子进程，父进程中 pid 为子进程 id，大于 0；子进程中 pid 为 0。首先考虑父进程的运行过程，if 条件句不成立，然后进入到 else 条件句，由于 wait(NULL) 的命令，使得父进程只有在子进程结束之后在可以继续向下执行，父进程暂停；子进程在创建的过程中拷贝了父进程的代码及变量，所以子进程中的 value 可以说是父进程中 value 的一个副本，在 if 条件句成立从而进行 value+=15 时，对父进程中 value 没有影响，最终子进程调用 return 0 后结束。此时，父进程开始执行，输出 value 的值，由于 value 没有改变，所以依旧是 5。

gdb 单步调试结果如下：

```
(gdb) l
1      #include <sys/types.h>
2      #include <stdio.h>
3      #include <unistd.h>
4      #include <stdlib.h>
5      int value = 5;
6      int main(){
7          pid_t pid=fork();
8          if(pid == 0){
9              value += 15;
10         } else if (pid > 0) {
(gdb) l
11             wait(NULL);
12             printf("PARENT: value = %d\n", value);
13             exit(0);
14         }
15         return 0;
16     }(gdb) break 7
Breakpoint 1 at 0x722: file os2.c, line 7.
(gdb) r
Starting program: /home/liuyh73/Desktop/a.out

Breakpoint 1, main () at os2.c:7
7          pid_t pid=fork();
(gdb) set follow-fork-mode child
(gdb) n
[New process 13350]
[Switching to process 13350]
main () at os2.c:8
8          if(pid == 0){
(gdb) n
9              value += 15;
(gdb) n
15         return 0;
(gdb) n
16     }(gdb) n
__libc_start_main (main=0x5555555471a <main>, argc=1, arg
    init=<optimized out>, fini=<optimized out>, rtd_fini=
    stack_end=0x7fffffffe008) at ../csu/libc-start.c:342
342     ../csu/libc-start.c: No such file or directory.
(gdb) set follow-fork-mode parent
(gdb) n
PARENT: value = 5
[Inferior 2 (process 13350) exited normally]
(gdb) q
```

红色方框中进程子进程与父进程的切换，并进程单步调试，最终可以看到 value 输出结果为 5。

3. 程序设计：

代码：

```
1. #include <sys/types.h>
2. #include <stdio.h>
3. #include <unistd.h>
4. int main(){
5.     pid_t pid1=fork();
```

```
6.     if(pid1==0)
7.         printf("Child1: b\n");
8.     else{
9.         pid_t pid2=fork();
10.        if(pid2==0)
11.            printf("Child2: c\n");
12.        else
13.            printf("Parent: a\n");
14.    }
15.    return 0;
16. }
```

运行结果:

```
(gdb) r
Starting program: /home/liuyh73/Desktop/a.out
Child1: b
Parent: a
Child2: c[Inferior 1 (process 13432) exited normally]
```

分析:

首先父进程 Parent 调用 fork, 创建子进程 Child1, 然后通过 if 条件句判断是 Parent 进程还是 Child1 进程:若是 Child1 进程,输出“Child: b” ;否则的话, 再次调用 fork 函数, 创建子进程 Child2, 利用同样的 if 条件句, 来判断 Parent 进程还是 Child2 进程: 若是 Child2 进程, 输出 “Child: c”; 否则的话, 输出 “Parent: a”。最终调用 return 0 结束程序。