

实验序号： 3



《UNIX/LINUX 编程环境》

实验报告

实验名称： Linux 系统进程编程

姓 名： 李昊唐

学 院： 工学院

专 业： 物联网工程

班 级： 1 班

学 号： 1995131017

指导教师： 彭凯

实验地址： 数学学院 416

实验日期： 2021 年 12 月 7 日

实验二 Linux 系统文件编程

一、实验目的

- (1) 理解 Linux 下产生新进程的方法（系统调用—fork 函数）。
- (2) 熟悉 wait，exit 等函数实现进程的同步功能。
- (3) 掌握管道通信方式

二、实验环境

实验配置

本实验所需的软硬件配置如表 1 所示。

配置	2.6 GHz 六核 Intel Core i7, 16 GB 2400 MHz DDR4, Intel UHD Graphics 630 1536 MB
硬件	MacBook Pro (15-inch, 2019)
系统	macOS 12.0.1
应用 软件	vi, sh

实验环境

本实验的环境为 Macintosh 机，如图 1 所示。



图 1 操作实验环境

三、实验原理

本实验相关函数

fork()函数创建一个新进程。

其调用格式为：int fork();

其中返回 `int` 取值意义如下：

正确返回：

等于 0：创建子进程，从子进程返回的 ID 值；

大于 0：从父进程返回的子进程的进程 ID 值。

错误返回：等于 -1：创建失败。

2、`wait()` 函数

`wait()` 函数常用来控制父进程与子进程的同步。在父进程中调用 `wait()` 函数，则父进程被阻塞，进入等待队列，等待子进程结束。当子进程结束时，会产生一个终止状态字，系统会向父进程发出 `SIGCHLD` 信号。当接到信号后，父进程提取子进程的终止状态字，从 `wait()` 函数返回继续执行原程序。

其调用格式为：

```
#include <sys/types.h>
```

```
#include <sys/wait.h>
```

```
(pid_t) wait(int *statloc);
```

正确返回：大于 0：子进程的进程 ID 值；

等于 0：其它。

错误返回：等于 -1：调用失败。

3、`exit()` 函数

`exit()` 函数是进程结束最常调用的函数，在 `main()` 函数中调用 `return`，最终也是调用 `exit()` 函数。这些都是进程的正常终止。在正常终止时，`exit()` 函数返回进程结束状态。

其调用格式为：

```
#include <stdio.h>
```

```
void exit(int status);
```

其中 status 为进程结束状态。

4、sleep（）函数

函数名：sleep

功 能：执行挂起一段时间

头文件：#include <unistd.h>

5、getpid（）函数功能描述：

getpid 返回当前进程标识

6. pipe（）

建立一无名管道。

系统调用格式

```
pipe(filedes)
```

参数定义

```
int pipe(filedes);
```

```
int filedes[2];
```

其中，filedes[1]是写入端，filedes[0]是读出端。

该函数使用头文件如下：

```
#include <unistd.h>
```

```
#include <signal.h>
```

```
#include <stdio.h>
```

read（）

系统调用格式

```
read(fd,buf,nbyte)
```

功能：从 fd 所指示的文件中读出 nbyte 个字节的数据，并将它们送至由指针 buf 所指示的缓冲区中。如该文件被加锁，等待，直到锁打开为止。

参数定义

```
int read(fd,buf,nbyte);
```

```
int fd;
```

```
char *buf;
```

unsigned nbyte;
write()

系统调用格式

read(fd,buf,nbyte)

功能：把 nbyte 个字节的数据，从 buf 所指向的缓冲区写到由 fd 所指向的文件中。如文件加锁，暂停写入，直至开锁。

参数定义同 read()。

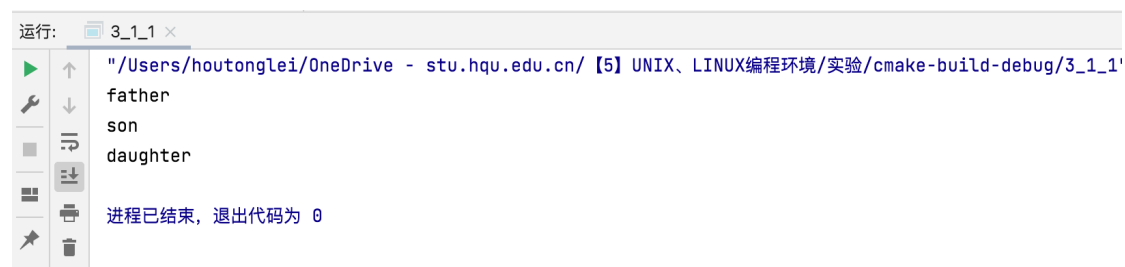
四、实验步骤和实验结果

1.编写一个 C 语言程序，实现在程序运行时通过系统调用 fork()创建两个子进程

(1) 使父、子、女三进程并发执行，父亲进程执行时屏幕显示“father”，儿子进程执行时屏幕显示“son”，女儿进程执行时屏幕显示“daughter”。

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

int main() {
    pid_t father = getpid(), son, daughter;
    son = fork();
    if(son == 0){
        puts("son");
        return 0;
    }
    daughter = fork();
    if(daughter == 0){
        puts("daughter");
        return 0;
    }
    if (getpid() == father) puts("father");
    return 0;
}
```



```

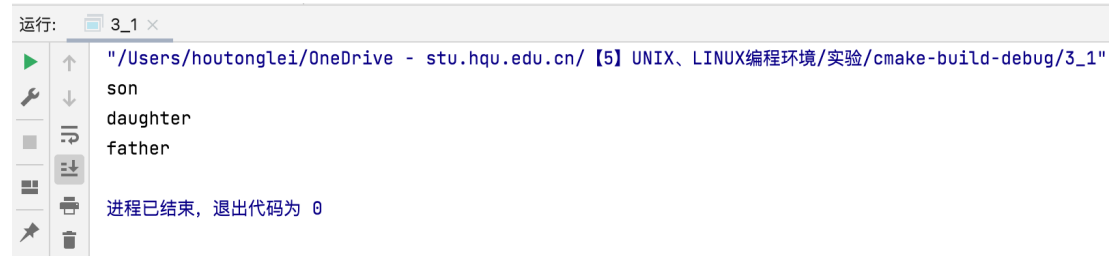
运行: 3_1_1 x
"/Users/houtonglei/OneDrive - stu.hqu.edu.cn/【5】UNIX、LINUX编程环境/实验/cmake-build-debug/3_1_1"
father
son
daughter
进程已结束，退出代码为 0
    
```

(2) 修改程序，在父、子进程中分别使用 `wait`、`exit` 等系统调用“实现”其同步推进。

```

#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>
#include <stdlib.h>

int main() {
    pid_t father = getpid(), son, daughter;
    son = fork();
    if(son == 0) {
        puts("son");
        exit(0);
    }
    daughter = fork();
    if(daughter == 0) {
        puts("daughter");
        exit(0);
    }
    wait(&son);
    wait(&daughter);
    if (getpid() == father) puts("father");
    return 0;
}
    
```



```

运行: 3_1 x
"/Users/houtonglei/OneDrive - stu.hqu.edu.cn/【5】UNIX、LINUX编程环境/实验/cmake-build-debug/3_1"
son
daughter
father
进程已结束，退出代码为 0
    
```

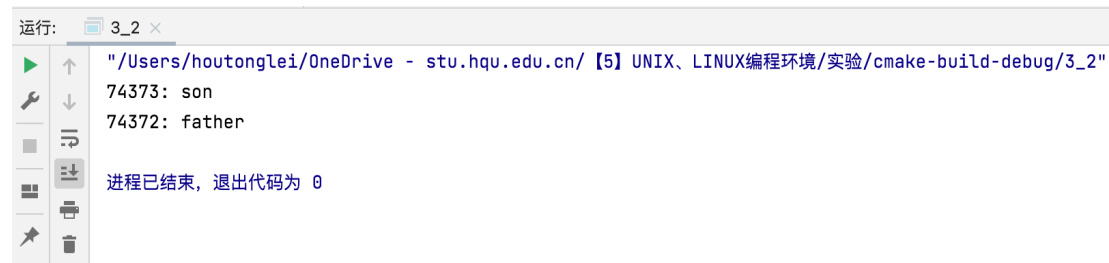
2.编写一个 C 语言程序，实现父进程等待子进程的功能。

要求：

- (1) 子进程创建完毕之后调用 `getpid()` 函数返回子进程 ID。
- (2) 子进程执行完毕后，调用 `sleep()` 函数睡眠 10s
- (3) 父进程用 `wait()` 函数等待子进程执行结束

```
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>
#include <stdlib.h>

int main() {
    pid_t father = getpid(), son;
    son = fork();
    printf("%d: ", getpid());
    if(son == 0) {
        puts("son");
        sleep(10);
        exit(0);
    }
    wait(&son);
    if (getpid() == father) puts("father");
    return 0;
}
```



3.创建命名管道 `myfifo`、命名管道 `yourfifo`；服务器端父进程往 `myfifo` 中写数据，客户端子进程从 `myfifo` 中读取服务器端父进程发送来的数据，客户端父进程往 `yourfifo` 中写数据，服务器端子进程从 `yourfifo` 中读取客户端父进程发送来的数据

服务端：

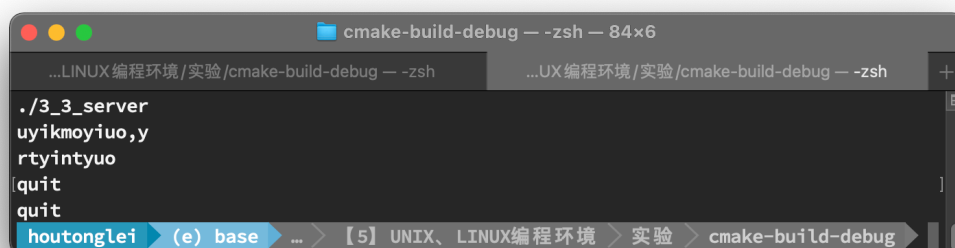
```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <fcntl.h>

int main() {
    int fd, fd1;
    pid_t child;
    char buf[1024], rbuf[1024];
    long bytes_read;

    mkfifo("myfifo", 0666);
    mkfifo("yourfifo", 0666);

    child = fork();

    if (child == 0) {
        while (strcmp(rbuf, "quit") != 0) {
            fd1 = open("yourfifo", O_RDWR);
            bytes_read = read(fd1, rbuf, sizeof(rbuf));
            rbuf[bytes_read] = '\0';
            printf("%s\n", rbuf);
            close(fd1);
        }
    } else {
        while (strcmp(buf, "quit") != 0) {
            fd = open("myfifo", O_RDWR);
            scanf("%s", buf);
            write(fd, buf, sizeof(buf));
            close(fd);
        }
    }
    return 0;
}
```

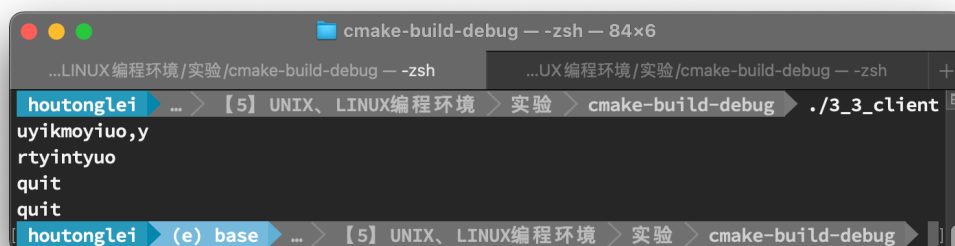


客户端:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>

int main() {
    int fd, fd1;
    pid_t child;
    char buf[1024], rbuf[1024];
    long bytes_read;

    child = fork();
    if (child == 0) {
        while (strcmp(buf, "quit") != 0) {
            fd1 = open("yourfifo", O_RDWR);
            scanf("%s", buf);
            write(fd1, buf, sizeof(buf));
            close(fd1);
        }
    } else {
        while (strcmp(rbuf, "quit") != 0) {
            fd = open("myfifo", O_RDWR);
            bytes_read = read(fd, rbuf, sizeof(rbuf));
            rbuf[bytes_read] = '\0';
            printf("%s\n", rbuf);
            close(fd);
        }
    }
    return 0;
}
```



五、实验总结

由于使用 lldb 对 fork 函数产生的子进程调试十分困难，且调试状态下的结果与运行状态下的结果不同，只能通过修改代码来调试。同时，相较 linux 系统，unix 系统下进程的管理有些许不同。