实验2　Linux进程控制与通信

1. 实验内容

(1) 编写一段程序（程序名为parent_child.c），使用系统调用fork()创建两个子进程，如果是父进程显示“Parent Process: A”，子进程分别显示“This is child1 (pid1 =xxxx)process: B”和“This is child1 (pid1 =xxxx)process: C”，其中“xxxx”分别指明子进程的pid号。

(2) 编写一段程序(程序名为comm.c)，父子进程之间建立一条管道，子进程向管道中写入“Child process 1 is sending a message!”，父进程从管道中读出数据，显示在屏幕上。

2. 实验要求

1. 将parentchild.c源程序，及程序执行结果写入实验报告；
2. 将fork()系统调用后内核的工作原理写入实验报告；
3. 将comm.c源程序，及程序执行结果写入实验报告；
4. 将Linux系统中管道通信的工作原理写入实验报告。

3. 实验步骤

(1) 以下是parent_child.c的源代码。

Listing 1: parent_child.c

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

int main(int argc, char *argv[])
{
    pid_t pid1, pid2;

    if ((pid1 = fork()) < 0) {
        fprintf(stderr, "创建子进程1失败\n");
        exit(1);
    } else if (pid1 == 0) {      /* 子进程1 */
        printf("This is child1 (pid1=%ld) process: B\n",
                (long) getpid());
    } else {                     /* 父进程 */
        printf("Parent Process: A\n");

        if ((pid2 = fork()) < 0) {
            fprintf(stderr, "创建子进程2失败\n");
            exit(1);
```

```
21          } else if (pid2 == 0) { /* 子进程2 */
22              printf("This is child2 (pid2=%ld) process: C\n",
23                      (long) getpid());
24          }
25      }
26
27      exit(0);
28  }
```

首先要知道fork()函数定义在<unistd.h>中。在主函数中，9–11行尝试创建一个子进程1并将其进程ID赋值给pid1。12–14行判断程序在执行子进程1，打印出相应信息；注意getpid()会返回一个pid_t类型的值，应将其转换为long输出。15–24是父进程，其中18–20行创建子进程2，21–24行判断是子进程2打印出相应信息。编译运行：

```
$ cc parent_child.c -o parent_child
$ ./parent_child
Parent Process: A
This is child1 (pid1=13221) process: B
This is child2 (pid2=13222) process: C
```

这里使用了一个小技巧，在创建了第一个子进程后，应该在父进程中再建立另外一个子进程，所以使用了嵌套的if语句。另外一种可行的方式是，使用wait()来等待子进程1结束后再fork()出来子进程2。
图1是编译和执行该程序以及下面程序的过程。

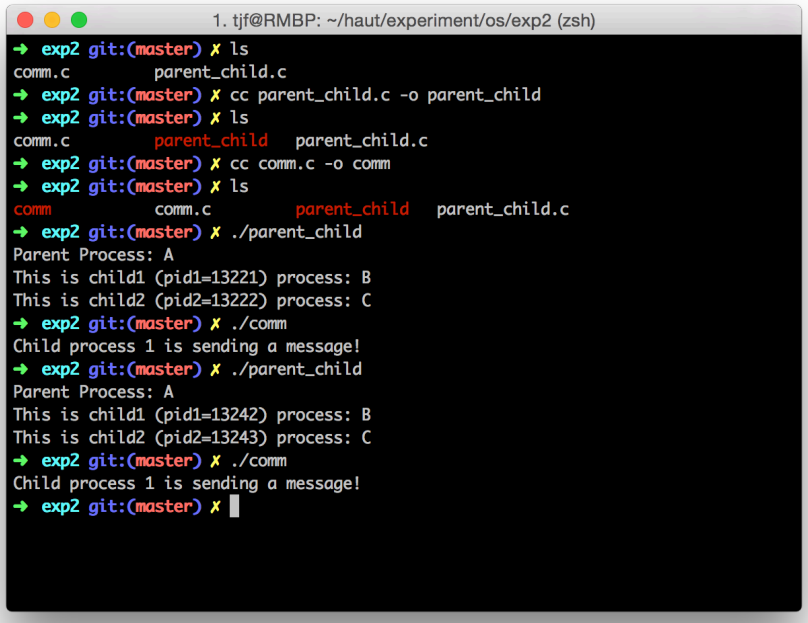

图 1: 实际执行过程

(2) 由fork()创建的是子进程。它被调用一次，返回两次。区别在于，子进程返回值是0，而父进程的返回值则是新建子进程的进程ID。子进程和父进程继续执行fork()调用之后的指令。子进程是父进程的副本，获得父进程的数据空间、堆和栈的副本。然而父进程和子进程并不共享这些存储部分，它们之间共享的是正文段。在实际实现中，操作系统通常并不完全复制其数据段和堆栈，而是采用了写时复制的技术来提高效率。

(3) 以下是comm.c的源代码。

Listing 2: comm.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define MAXLINE 512

int main(int argc, char *argv[])
{
    int n;
    int fd[2];
    pid_t pid;
    char *s = "Child process 1 is sending a message!\n";
    char line[MAXLINE];

    if (pipe(fd) < 0) {
        fprintf(stderr, "创建管道失败\n");
        exit(1);
    }

    if ((pid = fork()) < 0) {
        fprintf(stderr, "创建子进程失败\n");
        exit(1);
    } else if (pid == 0) {      /* 子进程 */
        close(fd[0]);
        write(fd[1], s, strlen(s));
    } else {                    /* 父进程 */
        close(fd[1]);
        n = read(fd[0], line, MAXLINE);
        write(STDOUT_FILENO, line, n);
    }

    exit(0);
}
```

pipe()也是在<unistd.h>中定义的。程序中，第6行定义了一个常量，表示一个文本行的最大长度。16-19行创建了一个管道，这个管道是从fd[0]读入，向fd[1]写出的。21-23行创建了一个子进程。24-26行，子进程关闭**读**描述符，向**写**描述符写出指定字符串，注意write()函数要求传入字符串的长度。27-31行是父进程，父进程关闭**写**描述符，从**读**描述符读取n个字节，写出到标准输出。这里使用了常量STDOUT_FILENO表示标准输出的文件号，默认是1。编译运行：

```
$ cc comm.c -o comm
$ ./comm
Child process 1 is sending a message!
```

这个程序同时用到了read()和write()用来读写文件描述符。

(4)