# 操作系统第三次上机实验

## 实验目标

实验目标
通过熟悉Linux系统中的管道通信机制，加深对进程间通信机制的理解
观察并体验并发进程间通信和协作的效果
练习利用无名管道进行进程间通信的编程和调试技术

## 实验内容

### 阅读并理解 02-lab3-demo-code.pdf 中的示例代码。

新建目录oslab3，进入oslab3目录，新建名为ppipe.c的C文件：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char* argv[]) {
    int pid;
    int pipe1[2];
    int pipe2[2];
    int x;
    if (pipe(pipe1) < 0) {
        perror("failed to create pipe1");
        exit(EXIT_FAILURE);
    }
    if (pipe(pipe2) < 0) {
        perror("failed to create pipe2");
        exit(EXIT_FAILURE);
    }
    pid = fork();
    if (pid < 0) {
        perror("failed to create new process");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // 子进程=>父进程：子进程通过pipe2[1]进行写
        // 子进程<=父进程：子进程通过pipe1[0]读
        // 因此，在子进程中将pipe1[1]和pipe2[0]关闭
        close(pipe1[1]);  // 关闭1的写
        close(pipe2[0]);  // 关闭2的读
        do {
            read(pipe1[0], &x, sizeof(int));
            printf("child %d read: %d\n", getpid(), x++);
            write(pipe2[1], &x, sizeof(int));
        } while (x <= 9);
        close(pipe1[0]);
        close(pipe2[1]);
    } else {
        // 父进程<=子进程:父进程从pipe2[0]读取子进程传过来的数
        // 父进程=>子进程:父进程将更新的值通过pipe1[1]写入，传给子进程
        // 因此，父进程会先关闭pipe1[0]和pipe2[1]端口
        close(pipe1[0]);
        close(pipe2[1]);
        x = 1;
        do {
            write(pipe1[1], &x, sizeof(int));
            read(pipe2[0], &x, sizeof(int));
            printf("parent %d read: %d\n", getpid(), x++);
        } while (x <= 9);
        close(pipe1[1]);
```

```
        close(pipe2[0]);
    }
    return EXIT_SUCCESS;

}
```

在oslab3中新建Makefile文件，内容如下:

```
srcs=ppipe.c
objs=ppipe.o
opts=-g -c
all:ppipe
ppipe: $(objs)
    gcc $(objs) -o ppipe
ppipe.o: $(srcs)
    gcc $(opts) $(srcs)
clean:
    rm ppipe *.o
```

在oslab3目录下，执行make命令

```
$ make
```

编译成功后，执行可执行程序ppipe

```
./ppipe
```

得到结果如下:

# 完成 03-lab3-assignments.pdf 中的上机任务

对于fock1.c

得到如下结果:

```
Child: value = 20
Child: value = 35
PARNET: value = 20
PARNET: value = 5
```

独立实验代码如下:

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

int fx(int x) {
    if (x == 1)
        return 1;
    return x * fx(x - 1);
}

int fy(int y) {
    if (y == 1 || y == 2)
        return 1;
    return fy(y - 1) + fy(y - 2);
}

int main() {
    pid_t pid_fx, pid_fy;
    int x, y;
    printf("Input x and y:\n");
    scanf("%d %d", &x, &y);

    // 创建第一个子进程计算 f(x)
    pid_fx = fork();
    if (pid_fx == 0) {
        printf("f(x) = %d\n", fx(x));
        exit(0);
    }

    // 创建第二个子进程计算 f(y)
    pid_fy = fork();
    if (pid_fy == 0) {
        printf("f(y) = %d\n", fy(y));
        exit(0);
    }

    wait(NULL);
    wait(NULL);

    printf("f(x, y) = %d\n", fx(x) + fy(y));

    return 0;
}
```

得到结果如下：

# 实验心得

对多线程编程有了进一步的了解，尤其是结合循环创建多个子进程的程序

管道是一种半双工通信方式，数据只能在一个方向上流动。在父进程和子进程之间使用管道时，通常父进程作为发送端，子进程作为接收端，或者相反。