哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验设计报告

开课学期：2022 年秋季

课程名称：操作系统

实验名称：XV6 与 UNIX 实用程序

实验性质：课内实验

实验时间：9.14 地点：T2210

学生班级：20 级 08 班

学生学号：200210231

学生姓名：王木一

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2022 年 9 月

一、 回答问题

1. 阅读 sleep.c，回答下列问题

（1） 当用户在 xv6 的 shell 中，输入了命令“sleep hello world\n"，请问在 sleep 程序里面，argc 的值是多少，argv 数组大小是多少。

argc = 3
argv 大小为 4

（2） 请描述上述第一道题 sleep 程序的 main 函数参数 argv 中的指针指向了哪些字符串，它们的含义是什么。

argv = [“sleep”, “hello”, “world\n”, 0]
argv[0] = “sleep” 需要执行的程序的名称
argv[1] = “hello” argv[2] = “world\n” 程序的参数（sleep 程序的参数应该是一个 int 数，此处参数错误）
argv[3] = 0 标志参数结束，argv 数组的最后一个元素必须为 0，这样程序才能正常执行

（3） 哪些代码调用了系统调用为程序 sleep 提供了服务？

```c
#include "kernel/types.h"
#include "user.h"

int main(int argc,char* argv[]){
    if(argc != 2){
        printf("Sleep needs one argument!\n"); //检查参数数量是否正确
        exit(-1);
    }
    int ticks = atoi(argv[1]); //将字符串参数转为整数
    sleep(ticks);              //使用系统调用 sleep
    printf("(nothing happens for a little while)\n");
    exit(0); //确保进程退出
}
```

如 user/sleep.c 中：
调用 printf()时，间接调用了 write() (第 6，11 行)
直接调用了 exit(),退出当前进程 (第 7，12 行)
调用 sleep(),暂停当前进程 (第 10 行)

2. 了解管道模型，回答下列问题

（1） 简要说明你是怎么创建管道的，又是怎么使用管道传输数据的。

创建管道：

1. 初始化 int 数组 p，用于保存指向管道两端的文件标识符（file descriptor）
2. 调用系统调用 pipe(), 将 p 数组作为参数传入

使用管道：

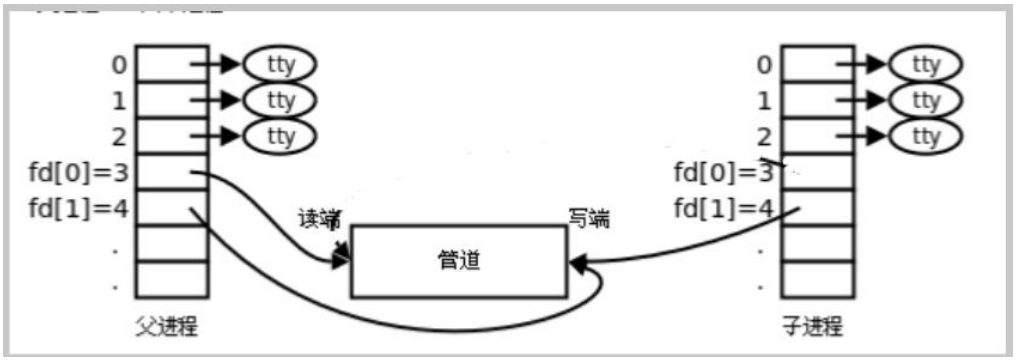
在管道一端的进程中调用 write()或 read(), 和读取普通的文件一样，需要注意的说当读（写）时，需要在当前进程中关闭写（读）的标识符

（2） fork 之后，我们怎么用管道在父子进程传输数据？

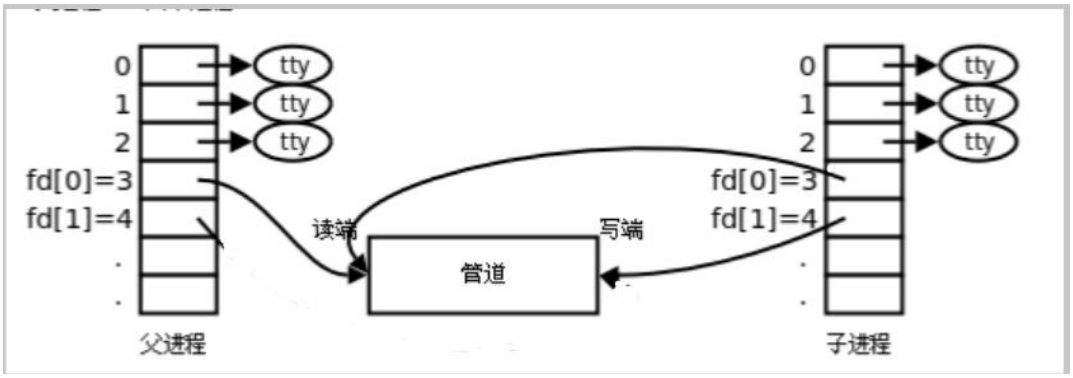
fork 之后父子进程都可以获得存有读写文件标识符的数组 p（前提已在父进程中调用了 pipe()）。在父子进程中分别使用读或写的系统调用即可传输数据。（write(p[1], buf, bytes), read(p[0], buf, bytes)）

（3） 试解释，为什么要提前关闭管道中不使用的一端？（提示：结合管道的阻塞机制）

管道两端的读写是阻塞的，即对于读端的进程，调用 read()函数时若管道中无数据，读端的进程将会阻塞，直至管道中被写入数据；对于写端的进程，调用 write()函数时若管道已被写满，写端的进程也会被阻塞，直至数据被读出。



考虑上图的情况，父进程为读进程，子进程为写进程，但父进程并未将自己进程中的写端关闭。当子进程写完，关闭子进程中的写端，父进程中的 read()并不会返回 0，告诉父进程子进程已经停止写入了，因为此时管道的写端仍有一个写入标识符打开，父进程就会被一直阻塞。



再考虑上图的情况：父进程为读进程，子进程为写进程，但子进程并未将自己进程中的读端关闭。当父进程读完，关闭父进程中的写端，子进程并不知道，因为管道读端仍有文件标识符打开，子进程仍然能写入数据，但当管道写满，子进程就会被一直阻塞。

“进程只持有管道的读出或写入端”的要求是为了避免进程被不正常的阻塞。

二、 实验详细设计

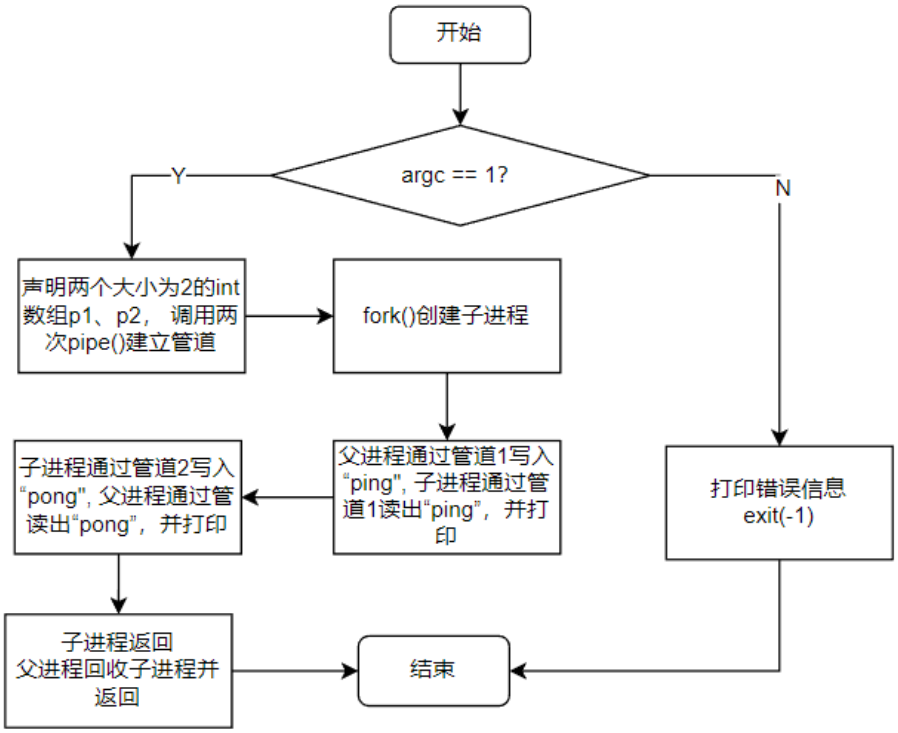
注意不要照搬实验指导书上的内容，请根据你自己的设计方案来填写

2.1 sleep

此程序已给出示例，故不再赘述。

2.2 pingpong

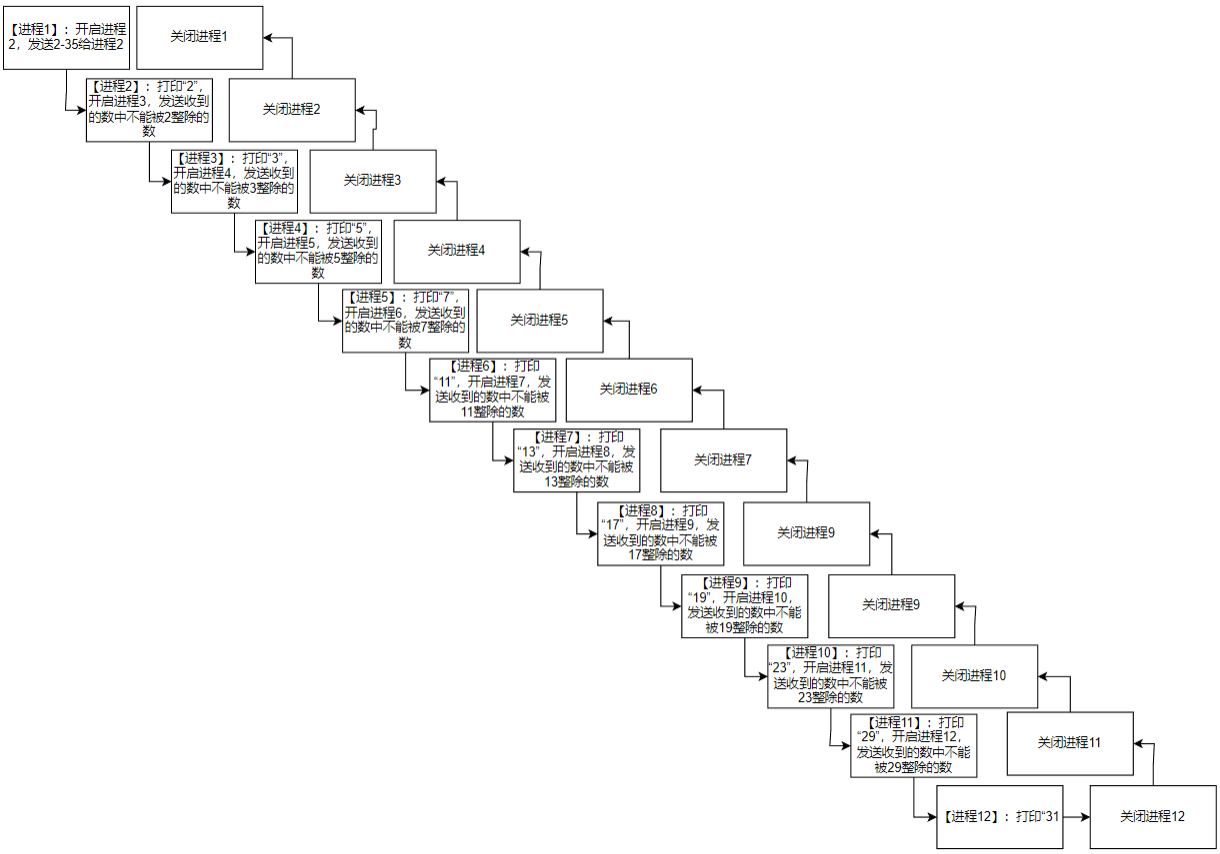
此题较为简单，开启两个相反方向的管道，分别完成“ping”和“pong”的传输即可。具体代码详见 pingpong.c，以下为流程图：



正如之前问题一.2.（3）父子进程在使用管道时要注意关闭不使用的一端

2.3 primes

受指导书“质数筛选的模型“启发，使用父子（祖孙）多个进程嵌套和管道实现质数筛选。以下为思路图：



祖辈进程要在子孙进程结束后才能结束，故使用 wait()方法实现。

具体代码详见 primes.c，以下以进程 11 和进程 12 的设计为例：

```
        if(fork() == 0){
            //process-11
            //print 29
            close(p10[1]);
            read(p10[0], num, sizeof(num));
            int out = num[0];
            fprintf(1, "prime %d\n", out);

            ///
            int p11[2];
            pipe(p11);
            if(fork() == 0){
                //process-12
                //print 31
                close(p11[1]);
                read(p11[0], num, sizeof(num));
                int out = num[0];
```

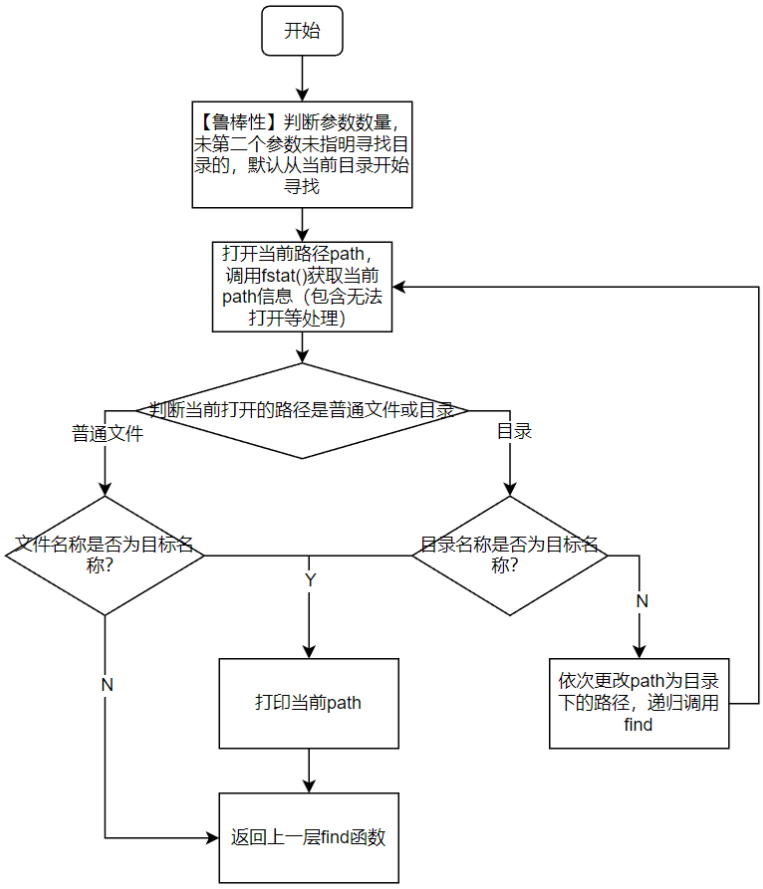
```
            fprintf(1, "prime %d\n", out);
            close(p11[0]);
            exit(0);
        }
        close(p11[0]);
        while(read(p10[0], num, sizeof(num))){
            if(num[0]%out != 0){
            //sent 31 to process-13
                write(p11[1], num, sizeof(num));
            }
        }
        close(p11[1]);
        wait(0);
        close(p10[0]);
        exit(0);
}
```

2.4 find

通过递归实现 find，同时参考了 ls 中对目录的读取和对文件名称的处理。

递归终止条件：①打开的是普通文件（同时包含找到和未找到两种情况）；②打开的是目录，但目录名称不是目标名称

具体代码详见 find.c，以下为流程图：



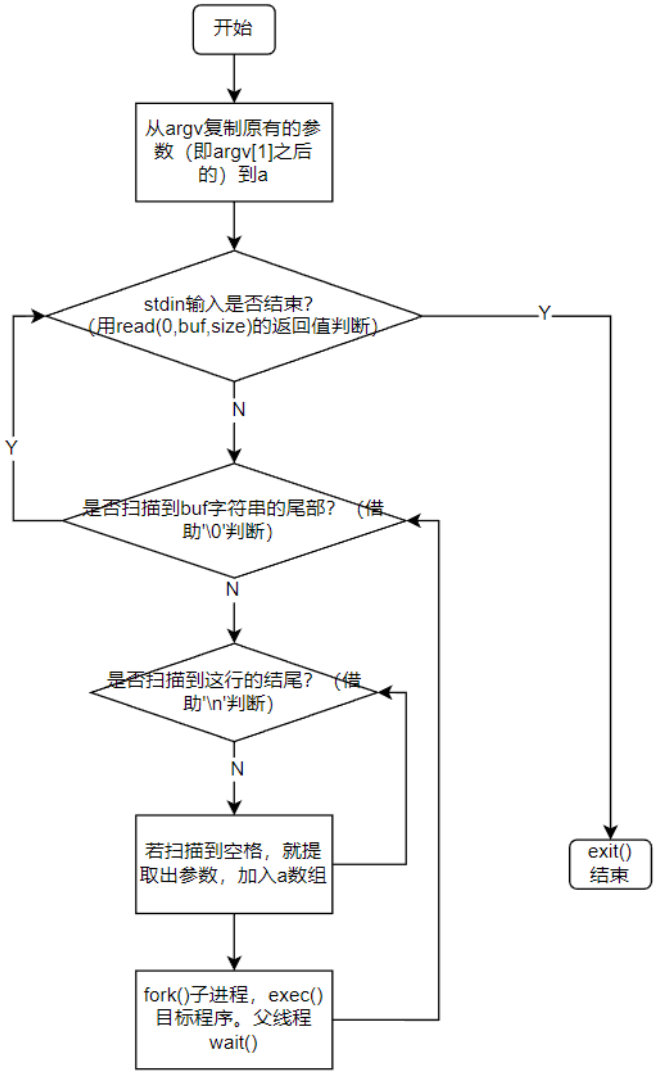
定义了 3 个函数：

- int main(int argc, char* argv[]):程序入口，包含输入合法性检测
- void find(char *path, char* target)（**参考自 ls.c**）
  1. 调用 fstat()获取文件信息
  2. 若打开的是普通文件，使用 strcmp()比对文件名和 target，若成功则打印 path，无论是否比对成功都要返回上一层递归调用
  3. 若打开的是目录，使用 strcmp()比对文件名和 target
     成功，打印 path，返回上一层递归调用
     失败，path 尾部添加‘/’，读取目录内容，使用 memmove()将读取出的内容添加至 path 尾部，递归调用 find。直至目录读取结束。
- char* fmtname(char* path) （**参考自 ls.c**）
  用于从 path 中提取出文件名称，如 path=”./a/b”，则返回值为“b”

2.5 xargs

主要思路：从 stdin 读取字符串 buf，从 buf 中提取出参数，fork()新进程，在新进程中调用 exec()执行程序

具体代码详见 xargs.c，以下为流程图：



1. 借助双指针和空格，提取出一行输入中的参数，借助'\n'判断此行时候读取结束
2. 由于可能从 stdin 中读取出多行，在读完一行后，判断其下一位是否为'\0'判断是否还有下一行。同时有几行输入就执行几次目标程序。

关键如下：

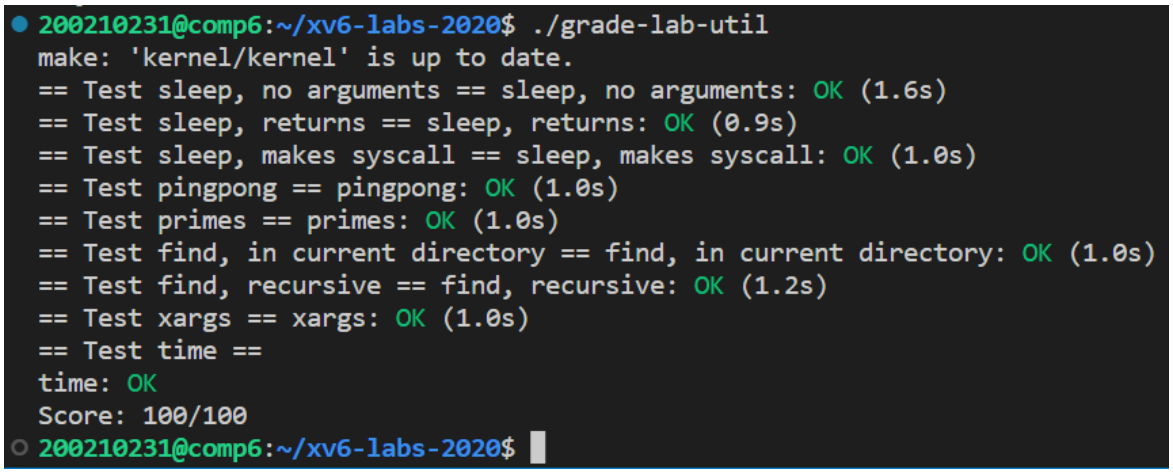
```
while(read(0, buf, 512) > 0){

        p = buf;
        q = buf;
        // extract args from stdin, 从 stdin 读入的是“一整块”（当管道发送端发送多行时,管道读出的是全部数据）， 借助\0 判断是否有多行， 空格分参数
        for(; p[0] != '\0';){
            i = argc - 1;
            for (; p[0] != '\n'; p++){
                if (p[0] == ' '){
                    *p++ = '\0';
                    a[i++] = q;
                    q = p;
                }
            }
            *p++ = '\0';
            a[i++] = q;
            q++;
            a[i] = 0;

            if (fork() == 0){
                exec(a[0], a);
            }
            wait(0);
        }
    }
```

三、 实验结果截图

请填写


200210231@comp6:~/xv6-labs-2020$ ./grade-lab-util
make: 'kernel/kernel' is up to date.
== Test sleep, no arguments == sleep, no arguments: OK (1.6s)
== Test sleep, returns == sleep, returns: OK (0.9s)
== Test sleep, makes syscall == sleep, makes syscall: OK (1.0s)
== Test pingpong == pingpong: OK (1.0s)
== Test primes == primes: OK (1.0s)
== Test find, in current directory == find, in current directory: OK (1.0s)
== Test find, recursive == find, recursive: OK (1.2s)
== Test xargs == xargs: OK (1.0s)
== Test time ==
time: OK
Score: 100/100
200210231@comp6:~/xv6-labs-2020$