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

实验设计报告

开课学期：2023 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：2023 年 9 月 15 日 地点：T2507

学生班级：智能强基-计算机

学生学号：210010101

学生姓名：房煊梓

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2023 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 的值是 3，argv 数组大小是 3。

（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。

第一题的 argv 数组大小为 3，有 3 个指针。
argv[0]指向字符串“sleep”，含义为命令名称 sleep。
argv[1]指向字符串“hello”，argv[2]指向字符串“world\n”。它们的含义为一行“hello world”。

（3） 哪些代码调用了系统调用为程序 sleep 提供了服务？

以下代码调用了系统调用为程序 sleep 提供服务：
exit(-1); 调用了 int exit(int status)。
sleep(ticks); 调用了 int sleep(int n)。
exit(0); 调用了 int exit(int status)。

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。

通过系统调用 pipe()创建管道。
通过关闭管道的写端，调用 read()函数，再关闭管道的读端来通过管道读数据；
通过关闭管道的读端，调用 write()函数，再关闭管道的写端来通过管道写数据。

（2） fork 之后，我们怎么用管道在父子进程传输数据？

首先通过 pipe()创建两个管道，第一个管道用于父进程写入“ping”，子进程读取并打印信息，第二个管道用于子进程写入“pong”，父进程读取并打印信息。
将 fork()的返回值赋给一个 int 类型的变量 ret，通过判断 ret 的值来确定父子进程。
若 ret 的值为 0，则为子进程，在第一个管道读取并打印信息，在第二个管道写入信息。
若 ret 的值大于 0，则为父进程，在第一个管道写入信息，在第二个管道读取并打印信息。

（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）

从满管道读取数据，当打开读出端将数据读出后管道才能变为空管道，而如果不提前关闭不使用的写入端，则会出现写入满管道的情况，造成管道阻塞。

同样地，根据管道阻塞机制，向空管道写入数据，当打开写入端将数据写入后管道才能变为满管道，而如果不提前关闭不使用的读出端，则会出现读取空管道的情况，造成管道堵塞。

根据以上分析，需要提前关闭管道中不使用的一端以避免管道阻塞。

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

1.pingpong 详细设计

(1)对题目进行分析，父进程将“ping”写入一个管道，子进程将其读出并打印信息，然后子进程将“pong”写入另一个管道，父进程将其读出并打印信息。因此需要两个不同的管道，系统调用 pipe()创建管道，它们的读出端和写入端分别对应 p1[0]、p1[1]和 p2[0]、p2[1]。

```
int p1[2];
//int pi1;
//int pi1=pipe(p1); //创建管道, p1[1]为写入端, p1[0]为读出端
pipe(p1);

int p2[2];
//int pi2;
//pi2=pipe(p2);
pipe(p2);
```

(2)读取和写入需要系统调用 read()和 write()，因此准备一个 buffer 用于读写。通过 fork()创建子进程。

```
char buffer[8]={};

int ret = fork();
```

(3) 通过判断 ret 的值来确定父子进程，若 ret 为 0，则为子进程，若 ret>0，则为父进程。

针对题目的要求，进程在对管道进行写操作时，先关闭读端，写入信息，再关闭写端；进行读操作时，先关闭写端，读取信息，打印信息，再关闭读端。因此对于“ping”和“pong”的具体设计如下：

第一个过程，父进程关闭管道 1 的读端，向管道 1 中写入“ping”，再关闭管道 1 的写端；子进程关闭管道 1 的写端，读取管道 1 中的信息并打印，再关闭管道 1 的读端。

第二个过程，子进程关闭管道 2 的读端，向管道 2 中写入“pong”，再关闭管道 2 的写端；父进程关闭管道 2 的写端，读取管道 2 中的信息并打印，再关闭管道 2 的读端。

```
if(ret==0){          //子进程

    close(p1[1]);    //关闭写端
    read(p1[0],buffer,4);          //读取
    printf("%d: received %s\n",getpid(),buffer); //打印信息
    close(p1[0]);    //关闭读端

    close(p2[0]);    //关闭读端
    write(p2[1],"pong",4);         //写入"pong"
    close(p2[1]);    //关闭写端

}else if(ret>0){     //父进程

    close(p1[0]);    //关闭读端
    write(p1[1],"ping",4);         //写入"ping"
    close(p1[1]);    //关闭写端

    close(p2[1]);    //关闭写端
    read(p2[0],buffer,4);          //读取
    printf("%d: received %s\n",getpid(),buffer); //打印信息
    close(p2[0]);    //关闭读端

}
```

2.find 详细设计

(1)对题目进行分析，需要在目录树中查找名称与字符串匹配的所有文件，输出文件的相对路径，并且命令的格式为“find path file_name”。整体参照 ls.c 来编写程序，设计了 main 函数，find 函数和 fmtname 函数。

(2)main 函数

首先检查参数的数量是否正确。

将第三个参数即命令输入的文件名保存在 file_name 当中，用于与找到的文件名进行比较；将第二个参数即 path 传入 find 函数开始查找。

```
int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Find needs two arguments!\n");//检查参数数量是否正确
        exit(-1);
    }
    file_name = argv[2];    //获取命令行输入的文件名
    find(argv[1]);          //从当前路径查找
    exit(0);    //确保进程退出
}
```

(3)fmtname 函数

输入路径 path，参照 ls.c 编写程序，返回该路径最后一个斜杠后的名称。（比如输入./a/b/c，则返回 c，若 c 是文件夹则返回的是文件夹名，若 c 是文件则返回的是文件名）

```
char *fmtname(char *path) {
    char *p;

    // Find first character after last slash.
    for (p = path + strlen(path); p >= path && *p != '/'; p--)
        ;
    p++;
    return p;   //文件名在最后一个斜杠后
}
```

(4)find 函数

整体基本按照 ls.c 的框架编写程序，并且使用递归允许 find 进入子目录进行查找。

首先判断能否打开文件，能否获取文件信息。这部分与 ls.c 一致。

```
void find(char *path) {
    char buf[512], *p;
    int fd;
    struct dirent de;
    struct stat st;

    if ((fd = open(path, 0)) < 0) {
        fprintf(2, "find: cannot open %s\n", path);
        return;
    }

    if (fstat(fd, &st) < 0) {
        fprintf(2, "find: cannot stat %s\n", path);
        close(fd);
        return;
    }
```

判断 st.type，若为文件，则将输入的 path 的最后一个斜杠后的名称与命令输入的文件名进行比较（利用 strcmp()进行比较），若相等则表示目标文件名在该路径下，故打印该路径 path。

```
switch (st.type) {
    case T_FILE:
        if (strcmp(fmtname(path), file_name) == 0) {  //比较找到的文件名和命令行输入的文件名，若相等则输出path
            printf("%s\n", path);
        }
        break;
```

若 st.type 为文件夹，则使用 while 语句，循环读这个文件夹里面的文件夹。当该文件夹里的文件数为 0 或者文件夹名为“..”或“.”，则退出循环，这符合题目“不要递归进入..和.”的要求。

若不是以上情况，则用 memmove 拼接出子目录路径 buf。再判断能否获取新路径的信息。若能，则使用 find 递归进入子目录查找。

```
        case T_DIR:
            if (strlen(path) + 1 + DIRSIZ + 1 > sizeof buf) {
                printf("find: path too long\n");
                break;
            }
            strcpy(buf, path);
            p = buf + strlen(buf);
            *p++ = '/';
            while (read(fd, &de, sizeof(de)) == sizeof(de)) {
                if (de.inum == 0||strcmp(de.name, "..") == 0||strcmp(de.name, ".") == 0)    //不要递归进入..和.
                    continue;
                memmove(p, de.name, DIRSIZ);
                p[DIRSIZ] = 0;
                if (stat(buf, &st) < 0) {
                    printf("find: cannot stat %s\n", buf);
                    continue;
                }
                find(buf);  //递归，find进入子目录查找
            }
            break;
    }
    close(fd);
}
```

三、 实验结果截图

请给出三个用户程序运行截图和 xv6 启动流程实验输出截图

```
210010101@comp1:~/xv6-oslab23-hitsz$ ./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 (0.9s)
== Test pingpong == pingpong: OK (1.1s)
== Test find, in current directory == find, in current directory: OK (1.0s)
== Test find, recursive == find, recursive: OK (1.2s)
Score: 60/60
```

```
210010101@comp1:~/xv6-oslab23-hitsz$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -drive
 file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0
kernel/start.c:55       [210010101] in start,init driver,interrupts and change mode
kernel/main.c:12        [210010101] enter main,init kernel
kernel/main.c:13
kernel/main.c:14        xv6 kernel is booting
kernel/main.c:15
kernel/proc.c:186       [210010101] enter userinit
kernel/proc.c:201       [210010101] copy initcode to first user process
kernel/main.c:35        hart 2 starting
kernel/main.c:35        hart 1 starting
init: starting sh
[210010101] start sh through execve
$
```