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

实验设计报告

开课学期：第三学年秋季学期

课程名称：操作系统

实验名称：XV6 与 Unix 应用程序

实验性质：课内实验

实验时间：10.8 地点：T2608

学生班级：计算机科学与技术 10 班

学生学号：190111005

学生姓名：倪洪达

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2018 年 12 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 argc 的值是多少，argv 数组大小是多少。

答：argc 的值为 3，argv 数组的大小是 3．

（2） 请描述 main 函数参数 argv 中的指针指向了哪些字符串，他们的含义是什么。

答：argv 的指针指向的是所有通过 shell 输入的字符串，不同的指令含义不同。

（3） 哪些代码调用了系统调用为程序 sleep 提供了服务？

答：exit(-1); sleep(ticks); exit(0);调用了 sleep 函数给程序 sleep 提供了服务。

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。

答：先声明一个长度为 2 的 int 类型的数组，如 fd[2]，然后用函数 pipe 创建管道。使用管道传输数据，一般是在父进程中使用 write 函数将想要传递的数据从管道写入端 fd[1]写入,在子进程中用 read 函数从管道读取端 fd[0]中获取数据。

（2） fork 之后，我们怎么用管道在父子进程传输数据？

答：在 fork 返回值为 0 的情况下（即子进程），使用 read 函数从管道读取端 fd[0]中读取数据，即 read 函数的第一个参数设置为 fd[0];否则（即父进程），使用 write 函数将待传递的数据从管道 fd[1]端写入，即 write 函数的第一个参数设置为 fd[1]。

（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）

答：以子进程不关闭管道为例，若在子进程退出之前未关闭管道写入端，则父进程可以继续写入管道，多次写入直到管道填充完毕以后，再写入的时候就会阻塞。

二、 实验详细设计

(1) pingpong

使用 pipe 创建两个管道，其中一个父进程向子进程写，另一个子进程向父进程写。fork()以后，子进程先读后写，父进程先写后读，父子进程均在读完以后立即打印。子进程代码如下（父进程类似）：

```
if(pid == 0)
/*child*/
{
    char* msg1 = "xxxx"; //先设置成一个任意值
    char* msg2 = "pong";
    close(fd1[1]);
    close(fd2[0]);
    read(fd1[0], msg1, sizeof(msg1));
    printf("%d: received %s\n", getpid(), msg1);
    write(fd2[1], msg2, sizeof(msg2));
    exit(0);
}
```

(2) primes

先将 2 至 35 这 34 个自然数放在一个数组里。然后对这个数组进行递归，递归函数的另一个参数即数组当前有的数。每一次递归先判断数组中是不是只有一个数，是则打印并返回，否则打印出当前数组的第一个数，然后创建一个子进程和管道，在当前的父进程中写入当前数组里的所有数，在当前的子进程去掉第一个数的倍数，将剩下的数重新按顺序赋值给原数组。子进程代码如下：

```
    int pid = fork();
    if (pid == 0)
    {
        int t, i = 0, j = 0, count = 0;
        close(fd[1]);
        while (i < n)
        {
            read(fd[0], &t, 2);
            if (t % p != 0)
            {
                a[j] = t;
                j++;
                count++;
            }
            i++;
        }
        trans(a, count);
        exit(0);
    }
```

(3) find

核心思路是在 ls.c 文件上进行修改。需要先设计一个 fmtname 函数用于把路径转化为文件名的字符串格式，和 ls 中略有不同，代码如下：

```
char* fmtname(char *path)
{
    static char buf[DIRSIZ + 1];
    char *p;

    // Find first character after last slash.
    for(p = path + strlen(path); p >= path && *p != '/'; p--);
    p++;
    memmove(buf, p, strlen(p) + 1);
    return buf;
```

```
}
```

然后修改 ls.c 中的 ls 函数为 find 函数，使得输入的文件名和已存在的文件路径对应上的时候，输出该路径。由于可能不止一个符合的文件名，并且待查找可能在较深的子目录中，需要递归进行查找。从 ls 中修改后的 switch 代码如下：

```
switch(st.type){
    case T_FILE:
        if (strcmp(fmtname(path), input) == 0)
            printf("%s\n", path);
        break;

    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 || de.inum == 1 || strcmp(de.name, ".") == 0 ||
strcmp(de.name, "..") == 0)
            continue;
        memmove(p, de.name, strlen(de.name));
        p[strlen(de.name)] = 0;
        find(buf, input);
    }
    break;
}
```

(4) xargs

从标准输入中读取行并为每行运行一次指定的命令，且将该行作为命令的参数提供。核心思路是对字符串的处理和对字符指针的理解。需要将原命令中的字符指针和后来运行输入的字符指针放到同一个字符指针数组内。第一步即从 shell 的输入中获取 echo 之后的字符串，代码如下：

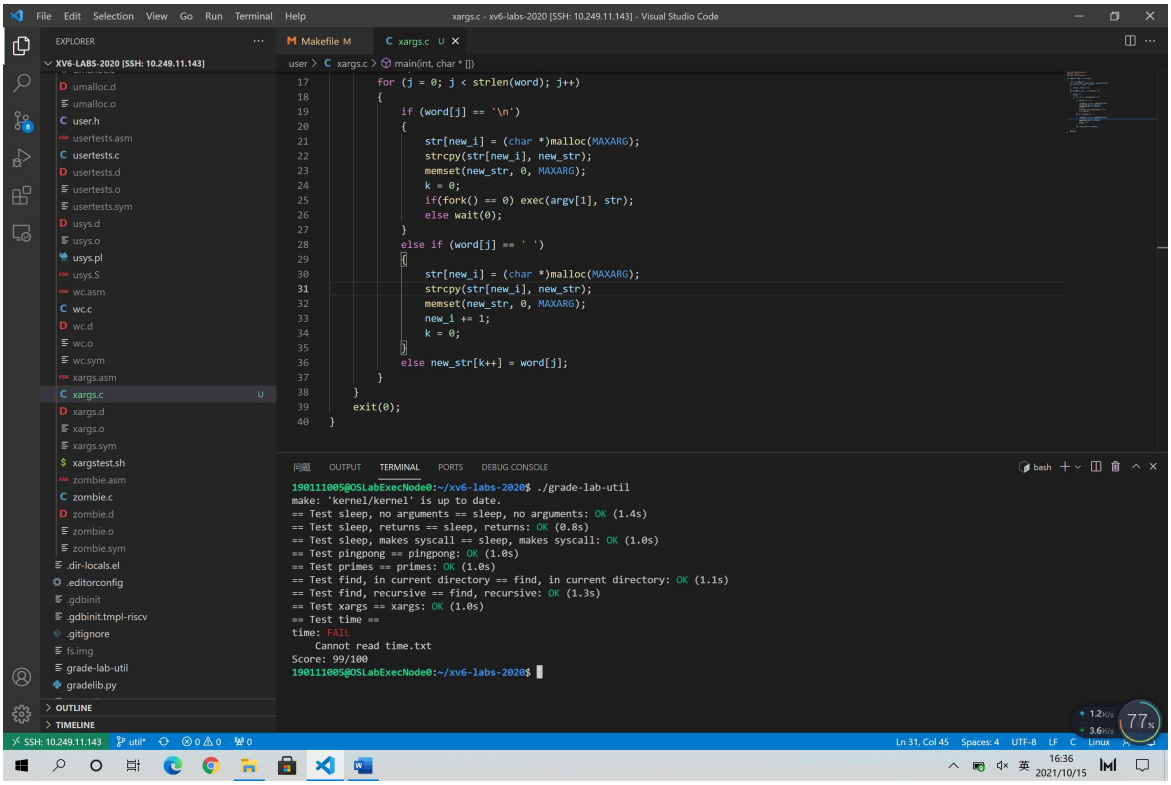
```
    for (i = 0; i < argc - 1; i++)
    {
```

```
        str[i] = argv[i + 1];
    }
```

其中，str 为新建的字符指针数组，也是后续子进程 exec 的时候需要的参数。后续需要从 shell 输入中不断地读取字符，主要分为三类，换行符、空格以及其他字符。其中换行符和空格在处理的时候，需要视作新输入了一条命令。于是使用 malloc 和 strcpy 来对原 str 数组进行扩增，并且用 new_str 来储存最新读入的字符串，代码如下：

```
    while(read(0, word, sizeof(word)) > 0)
    {
        new_i = i;
        k = 0;
        for (j = 0; j < strlen(word); j++)
        {
            if (word[j] == '\n')
            {
                str[new_i] = (char *)malloc(MAXARG);
                strcpy(str[new_i], new_str);
                memset(new_str, 0, MAXARG);
                k = 0;
                if(fork() == 0) exec(argv[1], str);
                else wait(0);
            }
            else if (word[j] == ' ')
            {
                str[new_i] = (char *)malloc(MAXARG);
                strcpy(str[new_i], new_str);
                memset(new_str, 0, MAXARG);
                new_i += 1;
                k = 0;
            }
            else new_str[k++] = word[j];
        }
    }
    exit(0);
```

三、 实验结果截图



文件 time.txt 按照指导书提示并未成功，除此之外测试均通过。