Lab0

姓名: 陈锐林, 学号:21307130148

2023 年 9 月 21 日

一、log_stdout.c:

1. 分析：

这题要求实现 read_stdin 和 log_stdout 两个函数，具体要求为，根据输入值 i，将标准输出重定向到 i.log；对于标准输入，先读入到 buf，再重定向到 i.log。

2. 实现：

实现：实现 log_stdout，分为两个步骤，一是根据输入得到 i 的值，这里将 i 的每一位转为 char；二是关闭标准输出，用 open 函数将 fd(1) 重定向到 i.log。实现 read_stdin，只要一位位读入到预先设置好的 buf 缓冲区，最后直接 printf 就到 i.log 了。

```c
int read_stdin(char* buf) {
    int bytesRead = 0;
    char c;
    while (read(0, &c, 1) > 0) {
        buf[bytesRead] = c;
        bytesRead++;
        if (bytesRead >= 1024) {
            fprintf(2, "Buffer overflow\n");
            return -1;
        }
    }
    printf("%s",buf);
    return 0;
}
```

(a) read_stdin

```c
int log_stdout(uint i) {
    char log_name[15] = "0.log";
    uint base = 1, i_temp;
    if (i != 0) {
        for (base = 0, i_temp = i; i_temp != 0; ++base, i_temp /= 10);
        for (uint base_temp = 0, i_temp = i; i_temp != 0; ++base_temp, i_temp /= 10) {
            log_name[base - base_temp - 1] = '0' + i_temp % 10;
        }
        strcpy(log_name + base, ".log");
    }
    close(1);
    if (open(log_name, O_CREATE | O_WRONLY) != 1) {
        fprintf(2, "log_stdout: open failed\n");
        return -1;
    }
    return 0;
}
```

(b) log_stdout

3. 运行效果：

```
$ log_stdout 6
Hello, this is first question in Lab0.
$ cat 6.log
Hello, this is first question in Lab0.
```

二、composites.c:

1. 分析：

这题要求补全 composites 和 sub_progress 两个函数。(这里对 sub_progress 进行了略微的修改)。sub_progress 实现思路为，每次取出输入的第一个数（已确保是素数），将其输出；然后遍历并去除其倍数，利用管道，不断将未考察的数 (考察：即已被找到的素数和其倍数) 作为当前父进程的输出和子进程的输入。

2. 实现：

在这里，通过 write 和 read 完成对管道的读写，调用 log_stdout 作重定向输出；每次都输入一个-1 在数组的末尾，作为哨兵，如果只剩-1 就退出程序。为了便于实现（确保输出结果的工整），让子进程先 sleep 一会再继续执行。composites 函数完成数组的初始化及导入和第一次子进程。需注意要等待子进程完成并且及时关闭不用的管道。

```c
void sub_process(int p_left[2]) {
    int prime,m;
    read(p_left[0], &prime, sizeof(prime)); // 读第一个数
    if(prime == -1) { // -1，代表所有数字处理完毕
        exit(0);
    }
    int p_right[2];
    pipe(p_right);
    if(fork() == 0) {
        close(p_right[1]); // 子进程只读不写
        close(p_left[0]); // 子进程不用
        sleep(8);
        sub_process(p_right); // 子进程以父进程的输出作为输入
    }else{
        close(p_right[0]); // 父只写不读
        printf("prime: %d\n", prime);
        while(read(p_left[0], &m, sizeof(m)) && m != -1) {
            if(m % prime != 0) {
                write(p_right[1], &m, sizeof(m));
            }
            else printf("composite: %d\n", m);
        }
        m = -1;
        write(p_right[1], &m, sizeof(m));
        wait(0);
        exit(0);
    }
}
```

(a) sub_progress

```c
void composites() {
    int p_right[2],i=10;
    pipe(p_right);
    if (log_stdout(i) < 0) {
        fprintf(2, "composites: log_stdout %d failed\n", i);
        exit(1);
    }
    if(fork() == 0) {
        close(p_right[1]);
        sub_process(p_right);
        exit(0);
    } else {
        close(p_right[0]);
        int i;
        for(i=2;i<=35;i++){
            write(p_right[1], &i, sizeof(i));
        }
        i = -1;
        write(p_right[1], &i, sizeof(i));
    }
    wait(0);
    exit(0);
}
```

(b) composites

3. 运行效果：顺序为从左往右，最后几个素数没截取

```
prime: 2
composite: 4
composite: 6
composite: 8
composite: 10
composite: 12
composite: 14
composite: 16
composite: 18
composite: 20
```

(a) part1

```
composite: 22
composite: 24
composite: 26
composite: 28
composite: 30
composite: 32
composite: 34
prime: 3
composite: 9
composite: 15
```

(b) part2

```
composite: 21
composite: 27
composite: 33
prime: 5
composite: 25
composite: 35
prime: 7
prime: 11
prime: 13
prime: 17
```

(c) part3

三、xargs.c:

1. 分析：

题目要求从标准输入读取多个参数并且执行，大致思路为利用 fork 出的子进程调用 exec 即可。

2. 实现：

由 hint 可知要注意换行符，以及可以直接调用的 MAXARG。具体实现时，先保存 argv 中的参数（注意索引），之后从标准输入小心地每次读入一个字符，碰到换行符了要特殊处理，fork 出子进程执行参数；否则就正常后移指针，当这个读入循环结束时，该函数的目标也达到了。

```
int main(int argc, char* argv[]) {
    if (argc < 2) {  //错误检验
        fprintf(2, "Usage: xargs command\n");
        exit(1);
    }
    char* args[MAXARG+1];
    int j = 0,i = 1;
    for (;i < argc; ++i) {  //保存参数
        args[j++] = argv[i];
    }
    char buf[512];  //buf存参数
    char *p = buf;
    while (read(0, p, 1) == 1) {
        if ((*p) == '\n') {  //special情况
            *p = 0;
            int pid;
            if ((pid = fork()) == 0) {
                args[j] = buf;
                exec(argv[1], args);  //exec执行
                fprintf(2, "exec %s failed\n", argv[1]);
                exit(0);
            }else{
                wait(0);
                p = buf;
            }
        } else {
            ++p;    //正常前进
        }
    }
    return 0;
}
```

3. 运行效果：取两个例子验证

```
hart 1 starting
hart 2 starting
init: starting sh
$ echo hello too | xargs echo bye
bye hello too
```

(a) example1

```
hart 2 starting
hart 1 starting
init: starting sh
$ find . b | xargs grep hello
hello
hello
hello
```

(b) example2