

Lab 0 基础

2023.9.19

实验说明

- 总共七个实验，基础、锁各占一周，剩下的实验两周，最后一周作为总结以及成绩评定。
- 基础、锁、系统调用、页表管理、Traps、COW、文件系统。
- 每次的实验报告在下一个实验之前发到助教邮箱中。

实验课群



群聊: oslab-2023秋季
群



该二维码 7 天内 (9月 25 日前) 有效，重新进入
将更新

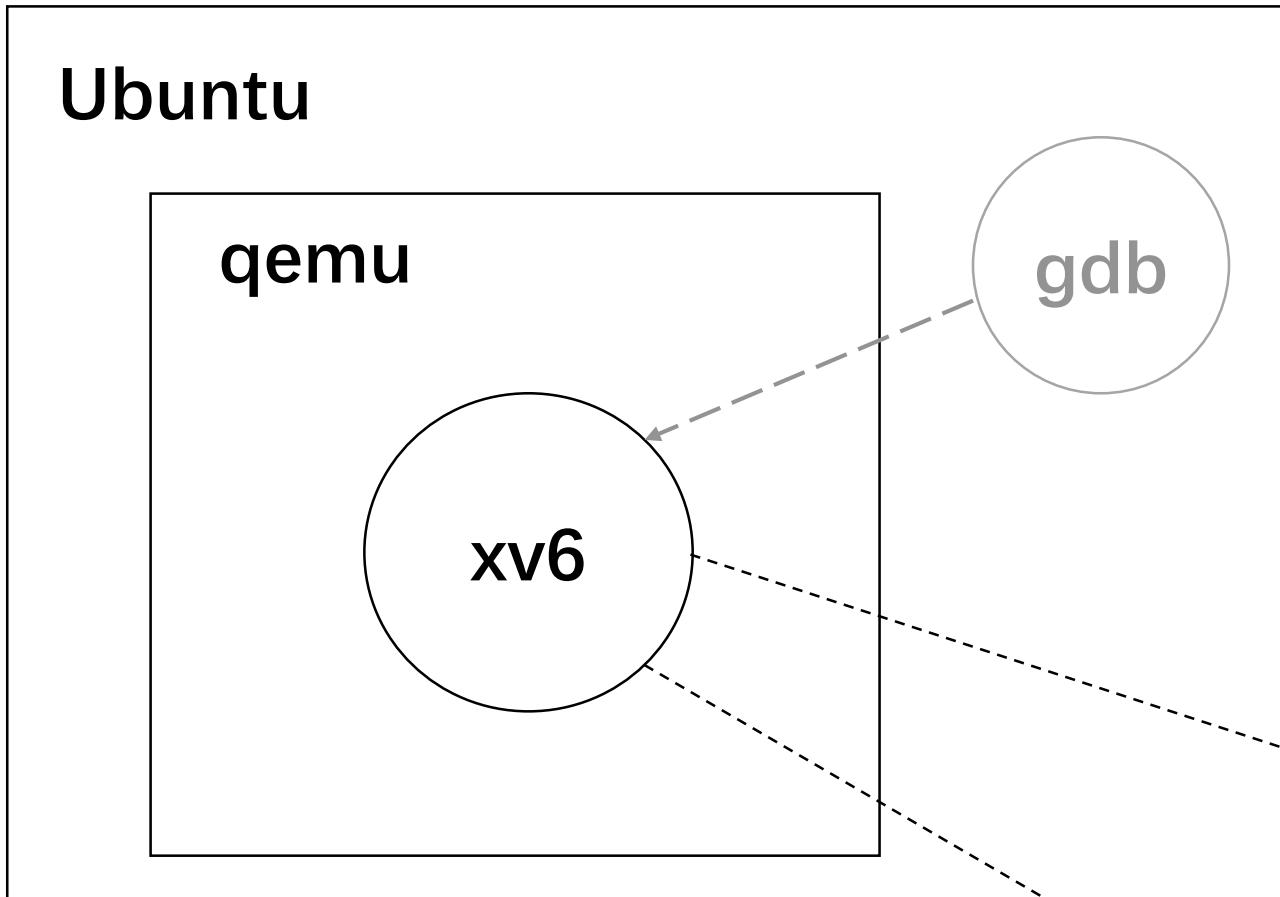
目录

- 实验目的
- 实验环境介绍
- 设置实验环境
- 常见系统调用函数
- I/O机制与Pipe的相关知识
- 练习和解答

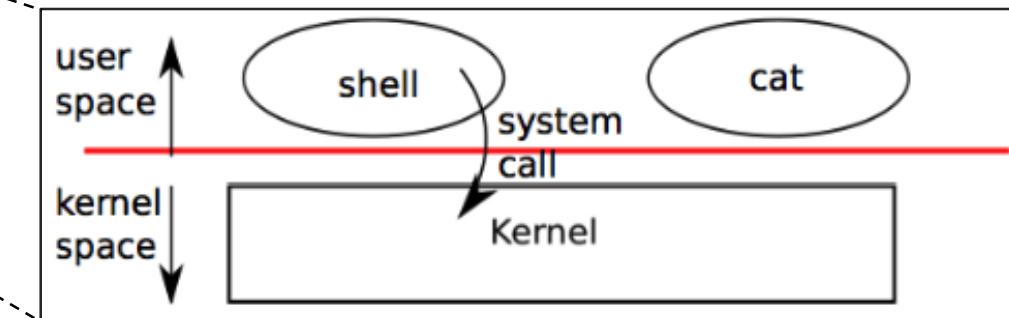
一、实验目的

- 了解xv6实验的环境
- 熟悉Linux命令行的使用
- 掌握常见系统调用函数的使用
- 了解内核的I/O机制
- 学习使用pipe进行进程间通信

二、实验环境介绍



- **qemu**：硬件模拟器
- **xv6**：实验操作系统
(类unix操作系统)
- **gdb**：调试工具



三、设置实验环境

- VirtualBox / VMware安装Ubuntu（课前准备）

- 在qemu硬件模拟器上安装xv6操作系统

打开终端

```
sudo apt-get install git build-essential gdb-multiarch qemu-system-misc gcc-riscv64-  
linux-gnu binutils-riscv64-linux-gnu qemu-system
```

- 编译并启动xv6系统

```
git clone git://g.csail.mit.edu/xv6-labs-2022 ~/Desktop/xv6-labs-  
2022  
cd ~/Desktop/xv6-labs-2022
```

make

- 之后每次使用**make qemu**命令即可在qemu中启动xv6系统
- 详见：[【腾讯文档】2023操作系统实验课准备工作](#)
- <https://docs.qq.com/doc/DZWxId0ZEaWFTTWNE>

三、设置实验环境

- xv6启动成功的效果

9月14日 15:31 ●

```
root@sherry-virtual-machine: ~/Desktop/xv6
objdump -s _ln > ln.asm
objdump -t _ln | sed '1,/SYMBOL TABLE/d; s/ .* / /; /$d' > ln.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o ls.o ls.c
ld -m elf_i386 -N -e main -Ttext 0 -o _ls ls.o ulib.o usys.o printf.o umalloc.o
objdump -t _ls > ls.asm
objdump -s _ls | sed '1,/SYMBOL TABLE/d; s/ .* / /; /$d' > ls.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o mkdir.o mkdir.c
ld -m elf_i386 -N -e main -Ttext 0 -o _mkdir mkdir.o ultb.o usys.o printf.o umalloc.o
objdump -s _mkdir > mkdir.asm
objdump -t _mkdir | sed '1,/SYMBOL TABLE/d; s/ .* / /'
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o rm.o rm.c
ld -m elf_i386 -N -e main -Ttext 0 -o _rm rm.o
objdump -s _rm > rm.asm
objdump -t _rm | sed '1,/SYMBOL TABLE/d; s/ .* / /'
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o sh.o sh.c
ld -m elf_i386 -N -e main -Ttext 0 -o _sh sh.o
objdump -s _sh > sh.asm
objdump -t _sh | sed '1,/SYMBOL TABLE/d; s/ .* / /'
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o stressfs.o stressfs.c
ld -m elf_i386 -N -e main -Ttext 0 -o _stressfs stressfs.asm
objdump -s _stressfs > stressfs.asm
cpu0: starting 1
cpu1: starting 0
objdump -t _stressfs | sed '1,/SYMBOL TABLE/d; s/ .* / /'
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o usertests.o usertests.c
ld -m elf_i386 -N -e main -Ttext 0 -o _usertests usertests.asm
objdump -s _usertests > usertests.asm
objdump -t _usertests | sed '1,/SYMBOL TABLE/d; s/ .* / /'
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o wc.o wc.c
ld -m elf_i386 -N -e main -Ttext 0 -o _wc wc.asm
objdump -s _wc > wc.asm
objdump -t _wc | sed '1,/SYMBOL TABLE/d; s/ .* / /'
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o zombie.o zombie.c
ld -m elf_i386 -N -e main -Ttext 0 -o _zombie zombie.asm
objdump -s _zombie > zombie.asm
objdump -t _zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /'
./mkfs.img README_cat_echo_forktest_grep_init_kill_ln_ls_mkdir_rm_sh_stressfs_usertests_wc_zombie
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
balloc: first 666 blocks have been allocated
balloc: write bitmap block at sector 58
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ 
```

四、常见系统调用函数

系统调用	描述
fork()	创建子进程
exit()	结束当前进程
wait()	等待子进程结束
exec(filename, *argv)	加载并执行文件
sleep(n)	睡眠n秒

参考：[Russ Cox, Frans Kaashoek, Robert Morris. xv6: a simple, Unix-like teaching operating system, chapter 1.1 pages 11](#)

四、常见系统调用函数

- fork 创建新进程

- 子进程的内存内容同创建它的进程（父进程）一样。
- fork 函数在父进程、子进程中都返回（一次调用两次返回）。
- 对于父进程它返回子进程的 pid，对于子进程它返回 0。

```
int pid;  
  
pid = fork();  
  
if(pid > 0) {  
    printf("parent: child=%d\n", pid);  
}  
  
pid = wait();  
printf("child %d is done\n", pid);  
  
} else if(pid == 0) {  
    printf("child: exiting\n");  
}
```

参考：[Russ Cox, Frans Kaashoek, Robert Morris. xv6: a simple, Unix-like teaching operating system, chapter 1.1 pages 11-12](#)

父进程 (pid >0) 进入分支；
并在wait()处等待子进程结束

①
子进程 (pid = 0) 进入分支；
并在exit()处结束返回

```
xv6 kernel is booting  
hart 1 starting  
hart 2 starting  
init: starting sh  
$ fork  
pcahrielndt:: exiting  
child=4  
child 4 is done  
$ █
```

四、常见系统调用函数

- exec读取内存镜像

- 注意exec并不会创建子进程，而是读取可执行文件的内存镜像，将其替换到当前进程内存空间。
- 所以为了使当前进程在exec后仍能正常运行，通常先fork一个子进程，然后在子进程中exec。

```
char *argv[3];  
argv[0] = "echo";  
argv[1] = "hello";  
argv[2] = 0;  
exec("/bin/echo",  
argv);  
  
printf("exec error\n");
```

大部分程序忽略第一个参数
(惯例为调用程序名)

将调用程序替换为 /bin/echo
这个程序传入argv参数组

由于内存空间被替换，exec成
功不会运行printf

```
xv6 kernel is booting  
hart 1 starting  
hart 2 starting  
init: starting sh  
$ exec  
hello  
$ █
```

参考：[Russ Cox, Frans Kaashoek, Robert Morris. xv6: a simple, Unix-like teaching operating system, chapter 1.1 pages 12](#)

五、I/O机制与Pipe

- I/O和文件描述符

```
char buf[512];
int n;

for(;;){
    n = read(0, buf, sizeof
buf);
    if(n == 0)           ← 读到0个字节表示
        break;           ← 读取结束
    if(n < 0){
        fprintf(2, "read
error\n");
        exit();
    }
    if(write(1, buf, n) != n){
        fprintf(2, "write
error\n");
        exit();
    }
}
```

系统调用	描述
read(fd, buf, n)	从文件中读 n 个字节到 buf
write(fd, buf, n)	从 buf 中写 n 个字节到文件

- 文件描述符 (fd) 是一个整数，代表了一个进程可以读写的被内核管理的对象；
- 按照惯例，进程从：
 - 文件描述符0读入（标准输入stdin）
 - 从文件描述符1输出（标准输出stdout）
 - 从文件描述符2输出错误（标准错误输出）
- 每一个指向文件的文件描述符都和一个偏移关联。

将数据从标准输入复制到标准输出
"cat"

```
xv6 kernel is booting
hart 2 starting
hart 1 starting
init: starting sh
$ r_w
hello
hello
```

五、I/O机制与Pipe

I/O重定向

(fork与文件描述符交叉使用)

```
char *argv[2];
```

```
argv[0] = "cat";
```

```
argv[1] = 0;
```

```
if(fork() == 0) {
```

```
    close(0);
```

```
    open("input.txt",  
          O_RDONLY);
```

复制父进程的文件描述符和内存

释放文件描述符0（标准输入）
使其可被open等重调用

为“input.txt”分配文件描述符0
(标准输入 指向 “input.txt”)

将“input.txt”（标准输入）内容
复制到标准输出

系统调用	描述
open(filename, flags)	打开文件，flags 指定读/写模式
close(fd)	关闭打开的 fd

- 新分配的文件描述符，永远都是当前进程的最小的、未被使用的文件描述符

```
xv6 kernel is booting  
hart 2 starting  
hart 1 starting  
init: starting sh  
$ echo "hello" > input.txt  
$ redirect  
$ "hello"
```

五、I/O机制与Pipe

- 使用Pipe完成进程间通信

```
int p[2]; // 创建pipe数组
char *argv[2]; // 执行参数
argv[0] = "wc";
argv[1] = 0;
pipe(p); // 创建一个管道，将读和写文件描述符分别放在p[0]和p[1]中
if(fork() == 0) {
    // 子进程
    close(0); // 释放之前其他进程的输入fd
    dup(p[0]); // 让fd0指向管道的读取端
    close(p[0]); // 关闭管道的读取端
    close(p[1]); // 关闭管道的写入端
    exec("/bin/wc", argv);
} else {
    // 父进程
    close(p[0]); // 关闭管道的读取端
    write(p[1], "hello world\n", 12); // 写入管道
    close(p[1]); // 关闭管道的写入端
}
```

系统调用	描述
dup(int fd)	返回指向与 fd 相同文件的新文件描述符
pipe(int p[])	创建管道，在 p[0] 和 p[1] 中放入读/写文件描述符

- 管道(pipe)为进程提供了一种通信方式。
- 管道是作为一对文件描述符公开给进程的小型内核缓冲区，一个用于读取，一个用于写入。

```
xv6 kernel is booting
hart 1 starting
hart 2 starting
init: starting sh
$ echo hello world | wc
1 2 12
```

实验练习

- 具体要求见 【腾讯文档】lab0 题
<https://docs.qq.com/doc/DZUtWRVRxYVBU3pj>
- log_stdout.c 和 composites.c 需要补全代码
- xargs.c 需要按照要求完成代码

实验练习

- 在user文件夹下创建xxx.c文件
- 在Makefile中补充实现的命令，如下
- 退出并运行make qemu，即完成编译，可在xv6中运行实现的命令

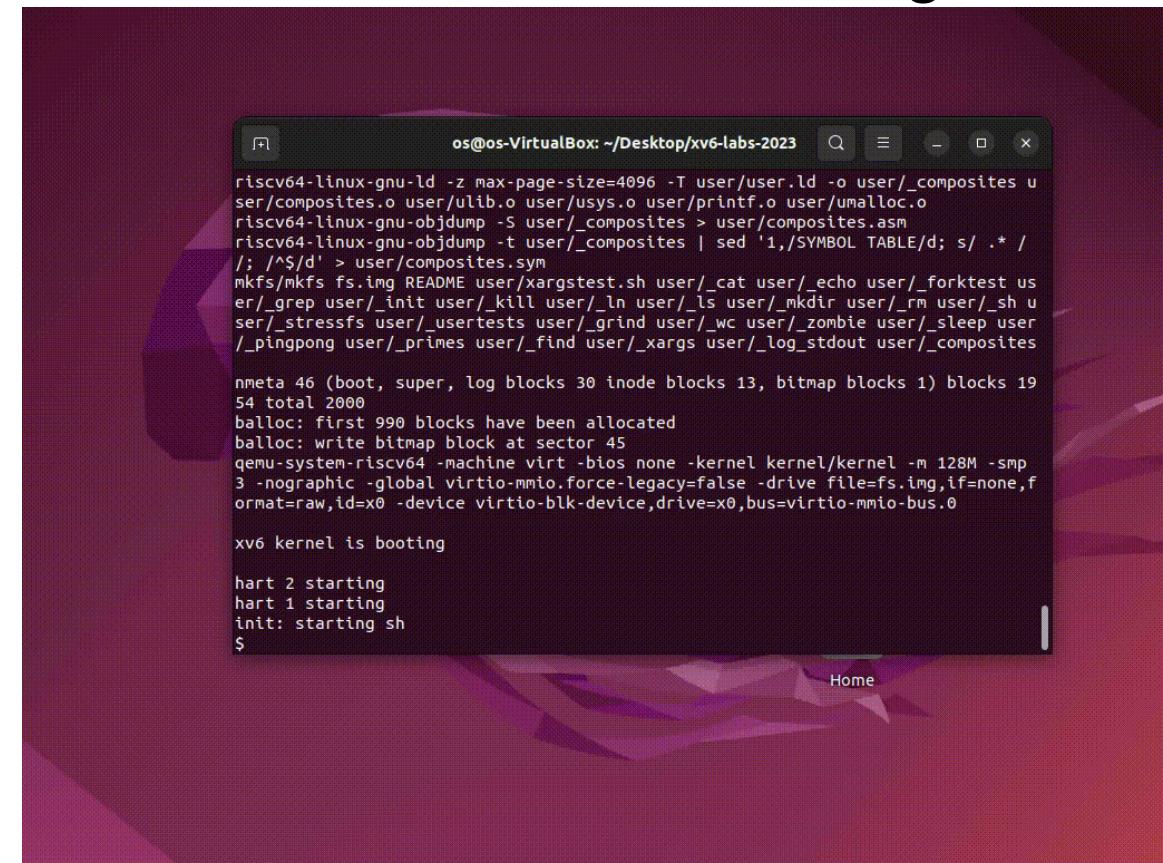
```
# Prevent deletion of intermediate files, e.g. cat.o, after first build, so
# that disk image changes after first build are persistent until clean. More
# details:
# http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
.PRECIOUS: %.o

UPROGS=\
    $U/_cat\
    $U/_echo\
    $U/_forktest\
    $U/_grep\
    $U/_init\
    $U/_kill\
    $U/_ln\
    $U/_ls\
    $U/_mkdir\
    $U/_rm\
    $U/_sh\
    $U/_stressfs\
    $U/_usertests\
    $U/_grind\
    $U/_wc\
    $U/_zombie\
    $U/_log_stdout\
    $U/_composites\
    $U/_xargs\
```

实验练习1 log_stdout.c

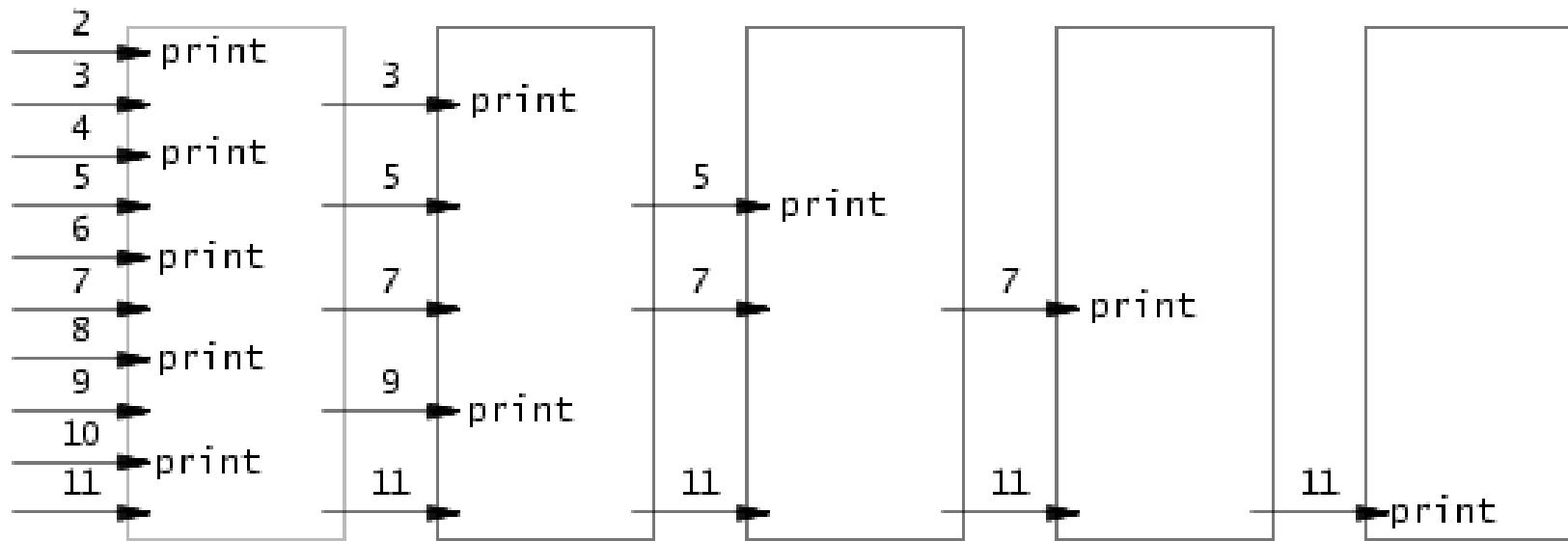
- 运行命令log_stdout i, i是uint类型
- 将stdout重定向到指定文件i.log中 (log_stdout)
- 读stdin到buf (read_stdin) , 打印buf, 打印内容被重定向到i.log

```
int main(int argc, char* argv[]) {
    if (argc != 2) {
        fprintf(2, "Usage: log_stdout number\n");
        exit(1);
    }
    if (log_stdout(atoi(argv[1])) != 0) {
        fprintf(2, "log_stdout: log_stdout failed\n");
        exit(1);
    }
    if (read_stdin(buf) != 0) {
        fprintf(2, "log_stdout: read_stdin failed\n");
    }
    printf(buf);
    exit(0);
}
```



实验练习2 composites.c

- 利用fork和pipe递归地实现流水线
- 流水线实现区分2-35之间的质数和合数
- 注意pipe只有需要的时候打开，且不需要时要及时关闭，不然会耗尽资源
- 用log_stdout函数将第i个子进程的stdout重定向到i.log
- 此图改自<https://swtch.com/~rsc/thread/>

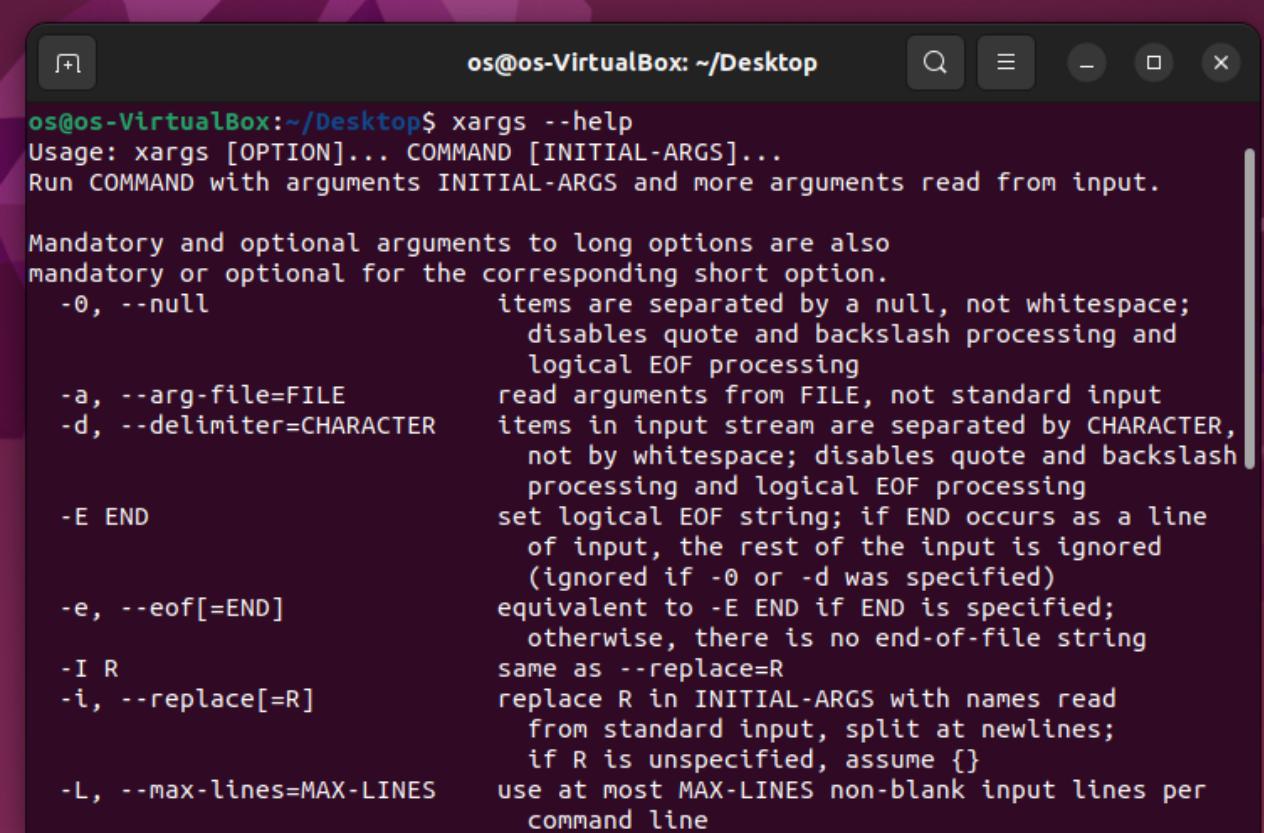


实验练习2 composites.c

```
riscv64-linux-gnu-ld -z max-page-size=4096 -T user/user.ld -o user/_composites user/composites.o user/ulib.o user/usys.o user/printf.o user/umalloc.o  
riscv64-linux-gnu-objdump -S user/_composites > user/composites.asm  
riscv64-linux-gnu-objdump -t user/_composites | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/composites.sym  
mkfs/mkfs fs.img README user/xargstest.sh user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_grind user/_wc user/_zombie user/_sleep user/_pingpong user/_primes user/_find user/_xargs user/_log_stdout user/_composites  
  
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 19  
54 total 2000  
balloc: first 990 blocks have been allocated  
balloc: write bitmap block at sector 45  
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0  
  
xv6 kernel is booting  
  
hart 1 starting  
hart 2 starting  
init: starting sh  
$
```

实验练习3 xargs.c

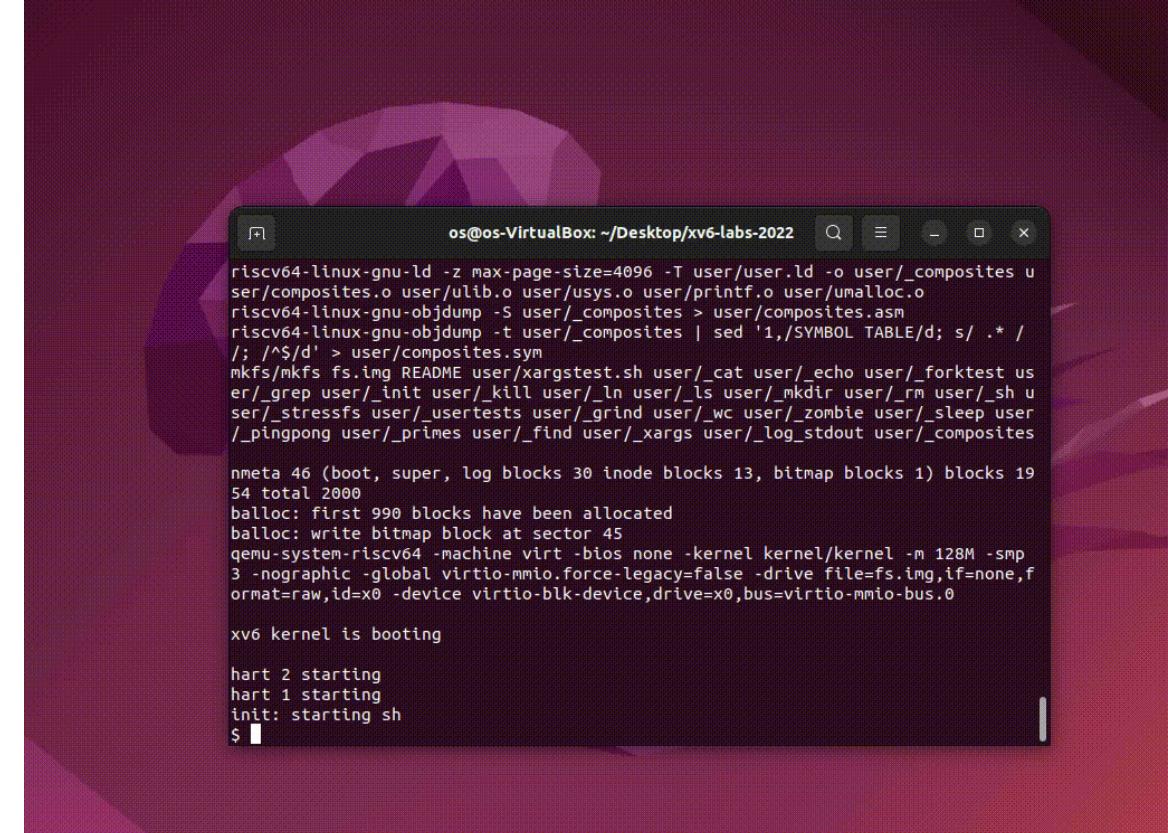
- 实现简易版xargs
- xargs command：将stdin作为command的参数，执行command
- 例如echo hello too | xargs echo bye等价于echo bye hello



```
os@os-VirtualBox:~/Desktop$ xargs --help
Usage: xargs [OPTION]... COMMAND [INITIAL-ARGS]...
Run COMMAND with arguments INITIAL-ARGS and more arguments read from input.

Mandatory and optional arguments to long options are also
mandatory or optional for the corresponding short option.

 -0, --null
                  items are separated by a null, not whitespace;
                  disables quote and backslash processing and
                  logical EOF processing
 -a, --arg-file=FILE
                  read arguments from FILE, not standard input
 -d, --delimiter=CHARACTER
                  items in input stream are separated by CHARACTER,
                  not by whitespace; disables quote and backslash
                  processing and logical EOF processing
 -E END
                  set logical EOF string; if END occurs as a line
                  of input, the rest of the input is ignored
                  (ignored if -0 or -d was specified)
 -e, --eof[=END]
                  equivalent to -E END if END is specified;
                  otherwise, there is no end-of-file string
 -I R
                  same as --replace=R
 -i, --replace[=R]
                  replace R in INITIAL-ARGS with names read
                  from standard input, split at newlines;
                  if R is unspecified, assume {}
 -L, --max-lines=MAX-LINES
                  use at most MAX-LINES non-blank input lines per
                  command line
```



```
riscv64-linux-gnu-ld -z max-page-size=4096 -T user/user.ld -o user/_composites user/composites.o user/ulib.o user/usys.o user/printf.o user/umalloc.o riscv64-linux-gnu-objdump -S user/_composites > user/composites.asm riscv64-linux-gnu-objdump -t user/_composites | sed '1,/SYMBOL TABLE/d; s/.*/;/; /[^/d' > user/composites.sym mkfs/mkfs fs.img README user/xargstest.sh user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_grind user/_wc user/_zombie user/_sleep user/_pingpong user/_primes user/_find user/_xargs user/_log_stdout user/_composites nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 19 54 total 2000 balloc: first 990 blocks have been allocated balloc: write bitmap block at sector 45 qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0 xv6 kernel is booting hart 2 starting hart 1 starting init: starting sh $
```

参考文献

- <https://pdos.csail.mit.edu/6.828/2022/tools.html>
- <https://pdos.csail.mit.edu/6.828/2022/labs/util.html>
- <https://pdos.csail.mit.edu/6.828/2022/xv6/book-riscv-rev3.pdf> **(建议自行阅读第一章)**

实验提交

- 提交到邮箱fduos2023lab@163.com：
 - 学号-姓名-oslab0.zip
 - log_stdout.c
 - composites.c
 - xargs.c
 - 学号-姓名-oslab0.pdf（中文报告）
 - 实现思路，测试结果
 - 实验中遇到的问题，如何思考并解决
- 截止日期：2023年9月25日23时
- 注意：请各位同学独立完成实验，参考代码需注明

总结与答疑

- 本次实验同学们需装好实验环境，然后根据系统进行一些简单练习。
- 接下来同学们可以开始实验了，有问题可随时提问。
- 下课前将机器退出系统，重新启动（但不要关机）。