

# 华南师范大学本科生实验报告

[Redacted]

院系：计算机学院

[Redacted]

年级：2022 级

[Redacted]

实验时间：

2024. 9. 8

实验名称：

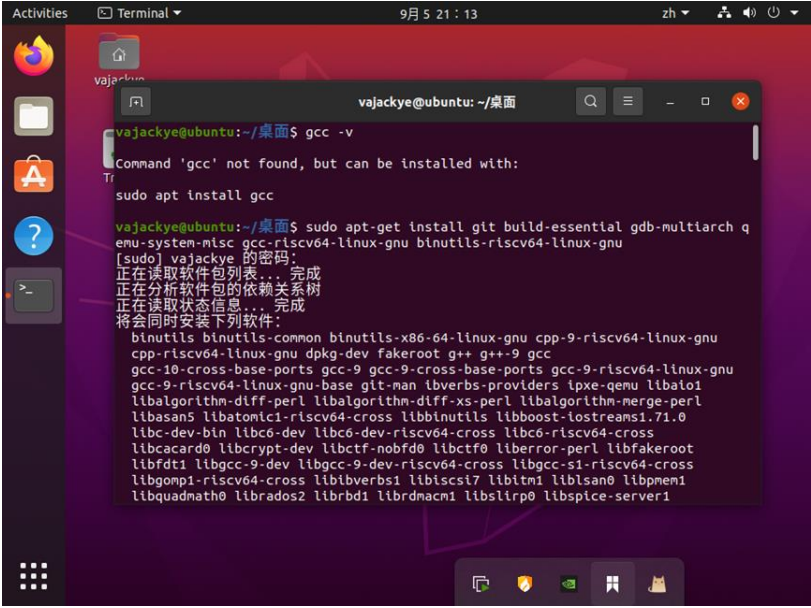
Lab1：实验平台搭建和几个练手程序

指导老师：

李丁丁

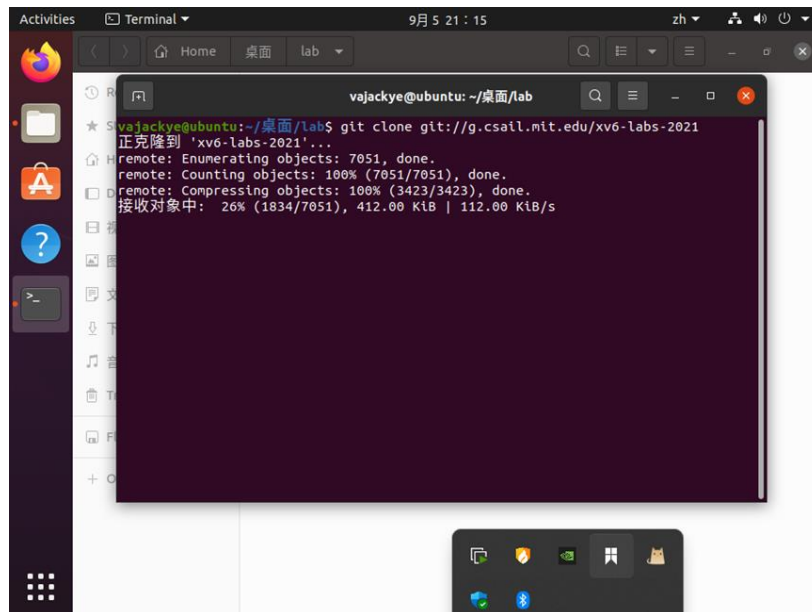
## 一. 实验环境搭建

- (1) 首先下载 VMware Workstation Pro, 和 ubuntu20.04.6 版本。然后配置和初始化 linux 环境。
- (2) 根据 qemu 官方的 ubuntu 的安装指令, 安装 qemu, git, gcc 等必要软件, 包。

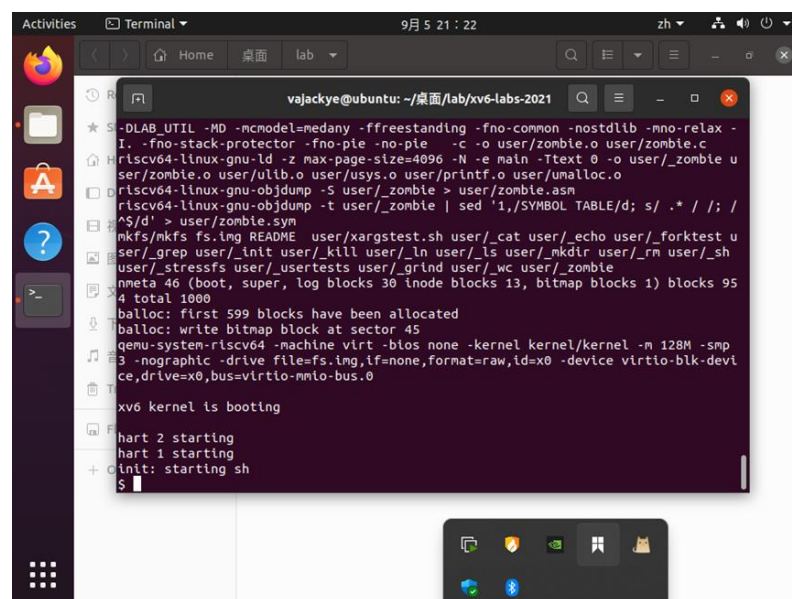


```
Activities Terminal 9月5 21:13 zh 人 音 电 网
vjackye@ubuntu: ~/桌面
vjackye@ubuntu:~/桌面$ gcc -v
Command 'gcc' not found, but can be installed with:
Try:
sudo apt install gcc
vjackye@ubuntu:~/桌面$ sudo apt-get install git build-essential gdb-multiarch q
emu-system-misc gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu
[sudo] vjackye 的密码:
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
将会同时安装下列软件:
binutils binutils-common binutils-x86-64-linux-gnu cpp-9-riscv64-linux-gnu
cpp-riscv64-linux-gnu dpkg-dev fakeroot g++ g++-9 gcc
gcc-10-cross-base-ports gcc-9 gcc-9-cross-base-ports gcc-9-riscv64-linux-gnu
gcc-9-riscv64-linux-gnu-base glibc-man libverbs-providers ipxe-qemu libaio1
libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
libasan5 libatomic1-riscv64-cross libbinutils libboost-iostreams1.71.0
libc-dev-bin libc6-dev libc6-dev-riscv64-cross libc6-riscv64-cross
libcacard0 libcrypt-dev libctf-nobfd0 libctf0 liberror-perl libfakeroot
libfdt1 libgcc-9-dev libgcc-9-dev-riscv64-cross libgcc-s1-riscv64-cross
libgomp1-riscv64-cross libibverbs1 libiscsi7 libitm1 liblsan0 libpnm1
libquadmath0 librados2 librbd1 librdnss1 libslirp0 libspice-server1
```

- (3) 配置完毕后, 建立一个文件夹, 并在文件夹内打开 terminal, 利用 git 命令, 拉取源代码。



(4) 使用命令: `make qemu` 进入 xv6 交互界面:



(5) 到此, 环境搭配完毕。

## 二. 实验 1---sleep.c

(1) 阅读实验文档, 得知要求睡眠时间为用户自己指定的时间  $t$  ( $t$  个  $t_0$ ,  $t_0$  为系统休眠单位), 并且由文档得知, 要使用系统封装函数 `int sleep(int n)`。

并且在写完 `sleep.c` 之后,要把文件放到 `../user/` 文件夹下,并配置 `makefile`

指令: `$U/_sleep`。最后,再在 `xv6` 根目录下 `make qemu` 进入 `xv6` 交互界面,敲 `sleep 5` 进行测试。

(2) 我们不妨先在 `makefile` 下添加指令:



(3) 然后按照题目要求,书写代码。根据 `book` 文档,了解 `argc` 和 `argv[]`,  
`exit()`,`write()` 等函数的功能后,便可开始书写代码。代码如下:

```
// 引入声明类型函数
#include "kernel/types.h"
#include "kernel/stat.h"
// 引入系统调用函数
#include "user/user.h"

/*
 * argc 是命令行总参数个数
 * argv[] 是 argc 个参数组成的字符数组, 其中第 0 个参数是程序全名,
 * 之后参数是命令行后跟的用户输入参数
 */
int
main(int argc, char* argv[]) {
    //根据 chapter1 文档得知, argc->argument==2
    if(argc != 2) {
```

```

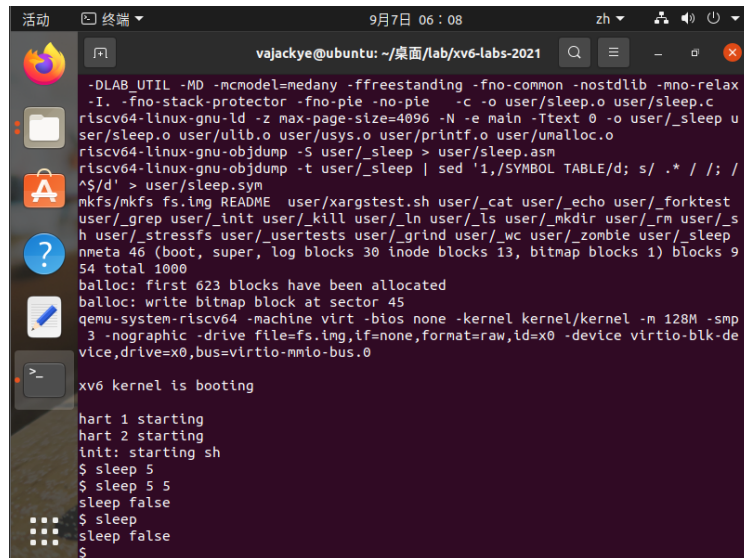
    /*
    fd: 文件描述符, 0_input, 1_output, 2_wr
    buf: buffer[]
    n: n byte
    */
    //write(fd, buf, int buf_byte_n) 输出错误
    write(2, "sleep false\n", strlen("sleep false\n"));

    //记得退出程序,!0 == error
    exit(-1);
}
if(time <= 0) exit(-1); //负数也不存在
//正确则执行: ,atoi 将 argv 内字符参数转换为 int
int time = atoi(argv[1]);
//调用系统 sleep 函数
sleep(time);
//ok to out
exit(0);
}

```

- (4) 该程序的思路为：根据文档，得知，argc 和 argv[] 概念，又第一个参数为程序全名，则我们要求输入一个时间 t，则共为输入两个参数。则判断 argc 是否为 2 个参数，false 则写出错误信息，并异常返回；若为两个参数，则先判断是否为正 t，若为负，则依旧异常返回；全都正确则将 argv[1] 转为整型后，调用 sleep()。

- (5) 结果如图：



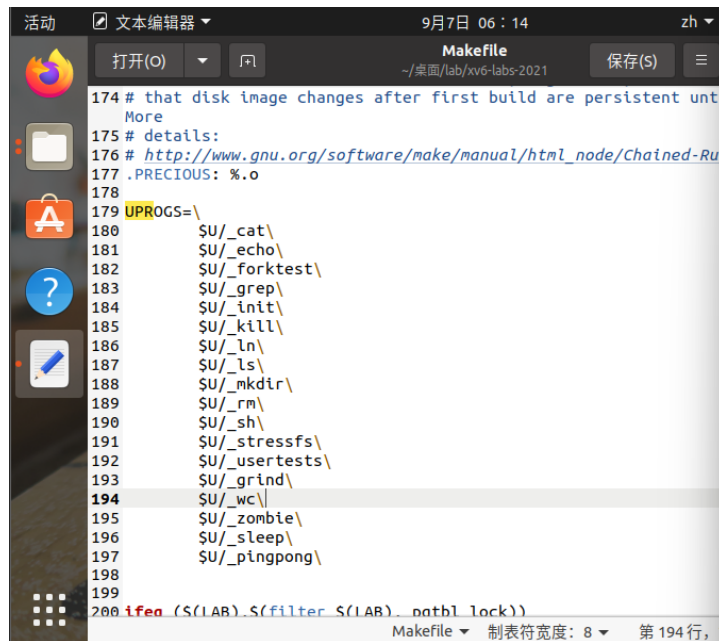
```
vajackye@ubuntu: ~/桌面/lab/xv6-labs-2021
-DLAB_UTIL -MD -mmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax
-I. -fno-stack-protector -fno-pie -no-pie -c -o user/sleep.o user/sleep.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_sleep u
ser/sleep.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-linux-gnu-objdump -S user/_sleep > user/sleep.asm
riscv64-linux-gnu-objdump -t user/_sleep | sed '1,/SYMBOL TABLE/d; s/ .* / /; /
^S/d' > user/sleep.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/_s
h user/_stressfs user/_usertests user/_grind user/_wc user/_zombie user/_sleep
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 9
54 total 1000
ballocc: first 623 blocks have been allocated
ballocc: write bitmap block at sector 45
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-de
vice,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ sleep 5
$ sleep 5 5
sleep false
$ sleep
sleep false
$
```

### 三. 实验 2---pingpong.c

- (1) 阅读实验文档，要求做父进程与子进程之间的响应操作。根据 book 的 chapter1 部分了解管道 `pipe()` 是单向的，`fork()` 是获取子进程的，上网查询两个函数的用法之后便可开始实验。
- (2) 我们先在 makefile 下添加指令：



(3) 开始书写代码。代码如下：

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

int main(int argc, char *argv[]) {
    int p1[2]; // 管道 p1, 父进程写, 子进程读
    int p2[2]; // 管道 p2, 子进程写, 父进程读
    // 创建两个管道, 双向
    pipe(p1);
    pipe(p2);
    // 创建子进程
    int pid = fork();
    if (pid == 0) { // pid == 0 子进程

        // 子进程操作
        close(p1[1]); // 关闭 p1 的写端 (子进程不写, 只读)
        close(p2[0]); // 关闭 p2 的读端 (子进程只写, 不读)

        char son[2];
        read(p1[0], son, 1); // 从 p1 的读端读取数据 (来自父进程)
        close(p1[0]); // 读取完成后关闭 p1 的读端
```

```

        // 打印"ping"
        printf("%d: received ping\n", getpid());
        // 向父进程发送响应 "pong"
        write(p2[1], "a", 2); // 向 p2 的写端写入数据（给父进程）
        close(p2[1]); // 发送完成后关闭 p2 的写端
    }
    else if (pid > 0) { // pid > 0 父进程
        close(p1[0]); // 关闭 p1 的读端（父进程只写，不读）
        close(p2[1]); // 关闭 p2 的写端（父进程不写，只读）
        // 父进程向子进程发送 "ping" 消息
        write(p1[1], "a", 2); // 向 p1 的写端写入数据（给子进程）
        close(p1[1]); // 发送完成后关闭 p1 的写端
        char father[2];
        read(p2[0], father, 1); // 从 p2 的读端读取数据（子进程）
        // 打印接收的消息 "pong"
        printf("%d: received pong\n", getpid());
        close(p2[0]); // 读取完成后关闭 p2 的读端
    }
    else { // 如果 fork 失败，打印错误信息
        printf("fork error\n");
    }
    exit(0); // 退出程序
}

```

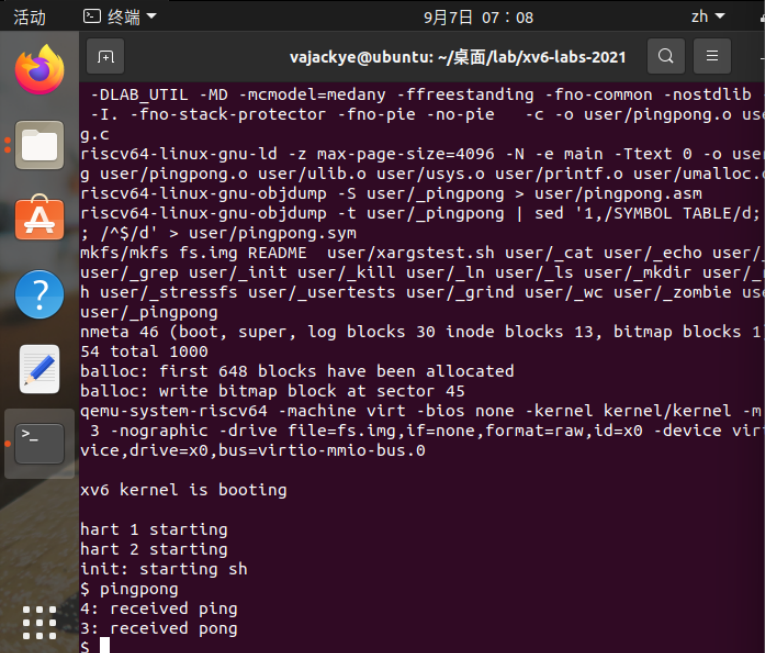
(4) 该程序的思路为：首先根据 book 的 chapter1 部分了解 `pipe` 和 `fork` 函数，搜寻用法后，编写。先创建两个管道（因为管道是单向的），一个给父进程写，子进程读；另一个给父进程读，子进程写。然后，`fork()` 创建子进程，获取 `pid`。

①接着判断 `fork()` 的返回值，如果 `pid == 0`，则表示当前运行的是子进程，关闭子进程不需要的管道部分，（关闭 p1 的写端和关闭 p2 的读端）。子进程从 p1 读取父进程发送的数据，并打印 "received ping"，然后向 p2 写入响应数据并关闭相应的管道；



②如果 `pid > 0`，则表示当前运行的是父进程，关闭父进程不需要的管道部分，（关闭 p1 的读端和关闭 p2 的写端）。父进程向 p1 写入 "ping" 数据，接着从 p2 读取子进程的响应数据，并打印 "received pong"，然后关闭管道

(5) 结果如图：



```
vajackye@ubuntu: ~/桌面/lab/xv6-labs-2021
-DLAB_UTIL -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib
-I. -fno-stack-protector -fno-pie -no-pie -c -o user/pingpong.o use
g.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o use
g user/pingpong.o user/ulib.o user/usys.o user/printf.o user/umalloc.o
riscv64-linux-gnu-objdump -S user/_pingpong > user/pingpong.asm
riscv64-linux-gnu-objdump -t user/_pingpong | sed '1,/SYMBOL TABLE/d;
/^$/d' > user/pingpong.sym
mkfs/mkfs fs.img README user/xargstest.sh user/_cat user/_echo user/_
user/_grep user/_init user/_kill user/_ln user/_ls user/_mkdir user/_
h user/_stressfs user/_usertests user/_grind user/_wc user/_zombie use
user/_pingpong
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1
54 total 1000
ballocc: first 648 blocks have been allocated
ballocc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m
3 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device vir
vice,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ pingpong
4: received ping
3: received pong
$
```

## 四. 实验 3---prime.c

(5) 阅读实验文档，要求筛选 2-35 的素数。 题目要求递归传递管道，每个递归只筛选一个素数。

(6) 我们先在 makefile 下添加指令：



(7) 开始书写代码。代码如下：

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

// 从管道中读取数据并筛选素数
void transform(int pfd) {
    int pid; // 子进程 ID
    int p[2]; // 新管道，用于子进程之间的通信

    int prm; // 存储读取的素数
    if (read(pfd, &prm, sizeof(int)) > 0) {
        // 打印当前素数
        printf("prime %d\n", prm);

        // 创建新的管道
        pipe(p);

        // 创建子进程
        pid = fork();
        if (pid < 0) {
            // 如果 fork 失败，打印错误信息并退出
            printf("fork error\n");
        }
    }
}
```

```

        exit(-1);
    } else if (pid == 0) {
        // 子进程
        close(p[1]); // 关闭管道写端
        transform(p[0]); // 递归调用，处理从管道中读取的数据
        exit(0); // 子进程完成后退出
    } else {
        // 父进程
        close(p[0]); // 关闭管道读端
        int k;
        while (read(pfd, &k, sizeof(int)) > 0) {
            if (k % prm != 0) {
                write(p[1], &k, sizeof(int)); // 写入管道
            }
        }
        // 关闭管道
        close(p[1]);
        close(pfd);
        // 等待子进程结束
        wait(0);
    }
} else {
    // 读取失败，关闭管道
    close(pfd);
}
}

int main() {
    int p[2]; // 定义主管道，0 输入，1 输出
    pipe(p); // 创建主管道
    int pid = fork(); // 创建子进程

    if (pid < 0) {
        // 如果 fork 失败，打印错误信息并退出
        printf("fork error\n");
        exit(-1);
    } else if (pid == 0) {
        // 子进程

```

```

        close(p[1]); // 关闭管道写端
        transform(p[0]); // 调用函数处理从管道中读取的数据
        exit(0); // 子进程完成后退出
    } else {
        // 父进程
        close(p[0]); // 关闭管道读端

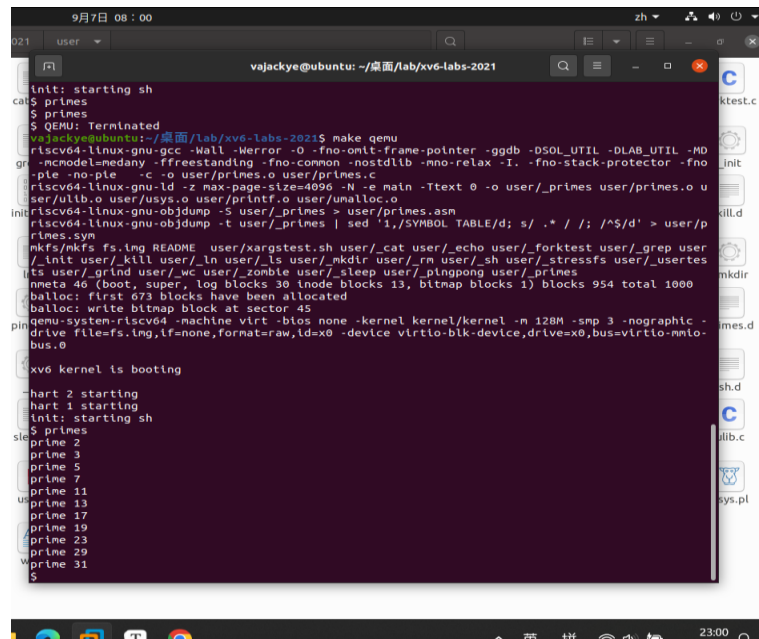
        // 向管道写入素数候选
        for (int i = 2; i <= 35; i++) {
            write(p[1], &i, sizeof(int)); // 将整数写入管道
        }

        // 关闭管道的写端
        close(p[1]);
        // 等待子进程结束
        wait(0);
    }
    exit(0);
}

```

(8) 该程序的思路为：创建一个管道，并 `fork()` 获取 `pid`，判断此时是子进程还是父进程在 `do`，利用埃氏筛优化算法，父进程用于筛选素数和筛掉该素数的倍数，并向新建的管道传递其余数，并等待子进程结束；子进程则递归调用传递管道的数据。直到所有数字筛选完毕。

(9) 结果如图：



```
9月7日 08:00
vjackye@ubuntu: ~/桌面/lab/xv6-labs-2021
init: starting sh
$ primes
$ primes
$ QEMU: Terminated
vjackye@ubuntu: ~/桌面/lab/xv6-labs-2021$ make qemu
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -DSOL_UTIL -DLAB_UTIL -MD
-mcmodel=medany -ffreestanding -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno
-plt -no-plt -c -o user/primes.o user/primes.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_primes user/primes.o u
ser/_lib.o user/_sys.o user/_printf.o user/_umalloc.o
riscv64-linux-gnu-objdump -S user/_primes > user/primes.asm
riscv64-linux-gnu-objdump -t user/_primes | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/p
rimes.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/_useres
ts user/_grind user/_wc user/_zombie user/_sleep user/_pingpong user/_primes
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 954 total 1000
ballocc: first 673 blocks have been allocated
ballocc: write bitmap block at sector 45
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
xv6 kernel is booting
hart 2 starting
hart 1 starting
init: starting sh
$ primes
prime 2
prime 3
prime 5
prime 7
prime 11
prime 13
prime 17
prime 19
prime 23
prime 29
prime 31
$
```

## 五. 实验心得

在本次实验中，完成了三个任务：`sleep()`、`pingpong()` 和 `primes()`，作为初学者，实验过程中通过阅读 xv6 的相关文档和源代码，我基本学会了操作系统的函数调用即代码编写。

在 `sleep()` 任务中，我实现了进程的休眠操作，理解了如何通过系统调用让进程进入休眠状态。在 `pingpong()` 任务中，我学习了 `pipe()` 和 `fork()` 的用法，学会了如何在父进程和子进程之间通过管道进行双向通信，实现了进程间的消息传递。在 `primes()` 任务中，加深了我对进程间通过管道通信的理解，特别是通过递归管道的方式筛选素数的实现，不仅让我对管道通信有了更深的掌握，还对递归和多进程的有了认识。

本次实验的难点，个人认为其实还是环境的搭建，刚开始搭建环境时，由于

是第一次接触 linux，遇到了很多问题，反复的重装，大约捣鼓了 2 个小时才算配置成功。之后的困难便是阅读全英文的实验书，在煎熬的用纯英文看完 chapter 后，才终于开始实验。