



操作系统实验

2022秋季

● 目录



● 本学期实验总体安排

实验课程共**24**个学时，**6**次实验课，**5**个实验项目，总成绩为**30分**。

实验项目	一	二	三	四	五
学时	4	4	4	4	8
实验内容	xv6与Unix实用程序	系统调用	锁机制的应用	页表	简单文件系统的设计与实现
上课时间	第3周	第5周	第8周	第10周	第12/14周
现场考核	第一次			第二次	第14周第三次

排课有冲突的同学可选择去其他班级补上，**作业提交在原班级**。

● 本学期课程考核

➤ 现场考核/验收(30%，第一次课占5%、第4次课占10%、最后一次课占15%)

- A: 自己独立完成，完成掌握本次实验，提问时能准确回答所有问题；
- B: 积极动手参与实验，但对内容还未完全掌握，少部分提问无法回答；
- C: 能回答出部分问题，动手实现过部分内容；
- D: 对于最基本的提问都无法准确回答，能看出来并未参与实验

➤ 代码及报告(70%)

- 实验1~4 提供运行实验设计报告；实验5 提交实验报告；
- xv6相关的实验项目，**仅需提交所有修改过的代码**；
- 实验5需**打包全部工程文件**提交。

● 本学期课程考核 | 附加题

完成5个基础实验后，感兴趣且时间充裕的，可以完成附加题，
争取加分（**实验分30分和平时分10分**）：

① MIT OS 6.S081/2020 Lab 1~10的要求：

- ① 提交能通过Lab 1~10 自测的完整项目代码；
- ② 提交完整的项目分析报告，包含对附加题MIT Lab 3、4、6、7、9、10 的**内容分析、设计方法、算法分析**等，格式不限；
- ③ 后续计划建设属于HITSZ的OS实验平台，请给出OS实验改进的意见和建议；
- ④ 由老师审核项目代码和分析报告，确认附加题完成情况达到要求。

在完成必做实验的基础上，选做了MIT Lab其他实验，但没能全部做完Lab1~10，也可以酌情加分！！！！

Labs ▾

xv6 ▾

R

Tools

Guidance

Lab Utilities

Lab System calls

Lab Page tables

Lab Traps

Lab Lazy allocation

Lab Copy on-write

Lab Multithreading

Lab Lock

Lab File system

Lab mmap

Lab network driver

● 课程平台

- **课程主页及指导书**地址: <http://hitsz-cslab.gitee.io/os-labs>
- **实验工具** 下载地址 (校内网) :
<http://10.249.14.14:8000/index.php/s/7vIEVZPKaMTIpi0>
- **实验提交** 地址 (校内网) : <http://10.249.12.98:8000/#/login>
- **Piazza**在线交流平台 (access code: comp3001) :
https://piazza.com/harbin_institute_of_technology_shenzhen/fall2022/comp3001

实验平台

➤ 可直接用的实验环境

远程实验环境

IP地址: **10.249.12.98**, 端口号: **6666**

➤ 自行部署的实验环境

1. 提供VirtualBox + openeuler的镜像, 直接导入镜像即可使用
2. 需自行下载、安装、编译所有工具链

相关工具

1. **虚拟机**: VirtualBox
2. **Linux发行版**: 由Linux内核、GNU工具、附加软件和软件包管理器组成的操作系统。
3. **RISC-V工具链**: 包括一系列交叉编译的工具, gcc, binutils, glibc等。
4. **QEMU**: 在X86上模拟RISC-V架构的CPU。

● 实验一任务 | 利用xv6系统调用实现5个Unix实用程序

➤ 实验一不修改XV6内核，而是编写应用程序，去使用操作系统。

◆ sleep(已给出源码)

◆ pingpong

◆ primes质数筛选

◆ find

◆ xargs

```
[cs@localhost xv6-labs-2020]$ ./grade-lab-util
make: 'kernel/kernel' is up to date.
== Test sleep, no arguments == sleep, no arguments: OK (1.2s)
== Test sleep, returns == sleep, returns: OK (1.0s)
== Test sleep, makes syscall == sleep, makes syscall: OK (0.9s)
== Test pingpong == pingpong: OK (1.4s)
== Test primes == primes: OK (0.9s)
== Test find, in current directory == find, in current directory: OK (1.1s)
== Test find, recursive == find, recursive: OK (1.1s)
== Test xargs == xargs: OK (1.9s)
== Test time ==
time: OK
Score: 100/100
[cs@localhost xv6-labs-2020]$
```


1 实验步骤 | 编译并运行xv6

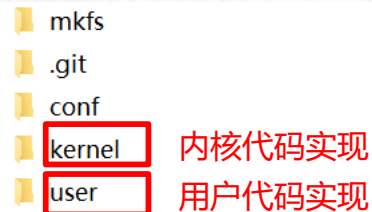
- 请参考实验指导书部署实验环境。
- 请clone最新代码到本地进行实验：

```
$ git clone https://gitee.com/hitsz-lab/xv6-labs-2020
```

- 每个实验项目都在不同的实验分支上完成，**请注意切换分支**，本实验需切换到**util分支**进行实现：

```
$ cd xv6-labs-2020  
$ git checkout util  
$ git branch
```

```
[cs@localhost xv6-labs-2020]$ git branch  
lazy  
lock  
master  
syscall  
* util  
[cs@localhost xv6-labs-2020]$
```



mkfs
.git
conf
kernel 内核代码实现
user 用户代码实现

.dir-locals.el
LICENSE
README
.cvsignore
.gdbinit
.gdbinit.tmpl-riscv
.gitignore
grade-lab-sh
grade-lab-util
gradelib.py
fs
Makefile

描述了编译、连接等规则，
增删文件时注意修改

1

实验步骤 | 编译并运行xv6

- 在代码总目录下输入 “make qemu” , 编译并运行xv6;
- 当可以看到 “init: starting sh” 的字样表示xv6已经正常启动, 此时在 “\$” 提示符后可输入xv6支持的shell命令。
- qemu退出方法:

先按 “Ctrl+a” 组合键, 接着全部松开,
再按下 “x” 键

```
[cs@localhost xv6-labs-2020]$ 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-devi
ce,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ ls
.                1 1 1024
..               1 1 1024
README          2 2 2059
xargstest.sh    2 3 93
cat              2 4 24216
echo            2 5 23032
forktest        2 6 13264
grep            2 7 27520
init            2 8 23776
kill            2 9 22984
ln              2 10 22824
ls              2 11 26408
mkdir           2 12 23136
rm              2 13 23120
sh              2 14 41936
stressfs        2 15 23976
usertests       2 16 148408
grind           2 17 38088
wc              2 18 25304
zombie          2 19 22368
sleep           2 20 22944
pingpong        2 21 23896
primes          2 22 24536
find            2 23 26696
xargs           2 24 24744
console         3 25 0
$
1 sleep  init
2 sleep  sh
$
```

输入ls命令

输入Ctrl + p显示进程信息

2

实验步骤 | 准备工作

✓ 在实验开始之前，强烈建议你先完成以下工作：

1. 熟悉常见命令的使用，如 `echo`、`xargs`、`find`。
2. 了解目录的使用。了解 `.`、`..`、`/` 分别表示什么，熟悉常见的目录操作命令，如 `mkdir`、`cd`。
3. 了解重定向的使用，重定向即命令中的 `<` 和 `>`，用于修改右侧命令的标准输入/输出。例如 `echo Hello world > file_a` 会将字符串 `Hello world` 输出至文件 `file_a`，而不是打印在终端。
4. 了解管道的使用。管道即命令中的 `|`，用于将左侧命令的标准输出传递给右侧命令的标准输入。
5. 了解常见系统调用的使用。如 `fork`、`exit`、`wait`、`open`、`close`、`read` / `write`、`pipe`、`dup`。

2 实验步骤 | 编译并运行sleep

- ✓ **Step1.** 阅读 *user/sleep.c* 文件, 理解代码和注释;
- ✓ **Step2.** 由于 *sleep.c* 为新增的用户程序文件, 请参考实验指导书修改 Makefile 文件;
- ✓ **Step3.** 编译 xv6 并运行 sleep.
- ✓ **Step4.** 回答指导书中的[相关问题](#)。

```
1 #include "kernel/types.h"
2 #include "user.h"
3
4 int main(int argc, char* argv[]){
5     if(argc != 2){
6         printf("Sleep needs one argument!\n"); //检查参数数量是否正确
7         exit(-1);
8     }
9     int ticks = atoi(argv[1]); //将字符串参数转为整数
10    sleep(ticks);                //使用系统调用sleep
11    printf("(nothing happens for a little while)\n");
12    exit(0); //确保进程退出
13 }
```

```
xv6 kernel is booting

virtio disk init 0
hart 1 starting
hart 2 starting
init: starting sh
$ sleep 10
Sleep 10
$ █
```

3

实验步骤 | 编写用户程序

根据[实验提示](#)，编写四个用户程序：

✓ Pingpong

✓ Primes

✓ Find

✓ xargs

1) pingpong

- 使用 `pipe()` 创建管道，详见[实验原理](#)；
- 使用 `fork()` 创建子进程，注意根据返回值，判断父子进程；
- 利用 `read()`，`write()` 函数对管道进行读写。
- 请在 `user/pingpong.c` 中实现。
- 修改 `Makefile`，将程序添加到 `UPROGS`。

2) primes

- 根据需求利用 `fork()` 创建子进程；
- 利用多个 `pipe()` 创建的管道在父子进程间进行数据的传输；
- `dup()` 可以用来复制文件句柄（管道的写入、读出端）：

例如，将"hello world"写入到标准输入。

```
fd = dup(1);
write(fd, "hello", 6);
write(fd, "world\n", 6); //此时fd1=1, 但字符串仍然会写入标准输出
```

3) find

- 可参照 `user/ls.c` 的逻辑实现；
- 使用递归允许 `find` 进入到子目录；
- 不要递归进入 `.` 和 `..`；
- 测试时需要创建新的文件和文件夹，可使用 `make clean` 清理文件系统，并使用 `make qemu` 再编译运行。

4) xargs

- xv6中的 `xargs` 基本功能演示：

示例1：

```
$ xargs echo good # 指定要执行的命令，echo，同时输入参数'good'
bye              # 换行后继续输入echo的参数'bye'
good bye         # 执行"echo good bye"，输出"good bye"
hello too        # 换行后输入参数'hello too'
good hello too   # 执行"echo good hello too"，输出"good hello too"
# 通过ctrl+D结束输入
$
```

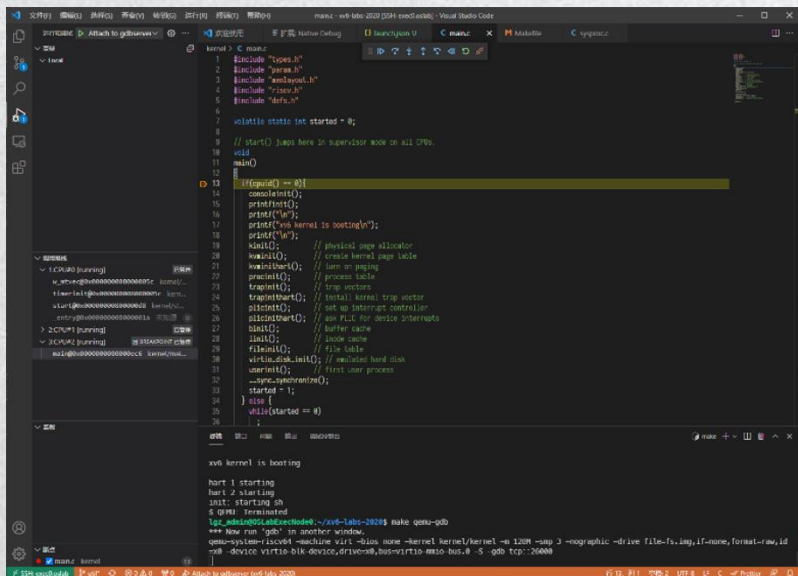
`echo` 命令用于将接收到的参数作为字符串输出。示例中，每输入一行字符串 `string`，程序就会执行一次 `echo good string`。用户输入 `ctrl+D` 的时候，`gets` 等函数返回为空。

需要注意的是，Linux中 `xargs` 具备 `-n` 选项，用于选择每次执行命令需要接收的参数数量，例如：

```
$ xargs -n2 echo good # 设置选项-n为2，表示接收两个参数（两行输入）；指定要执行的命令，echo，并输入参数'
bye                  # 换行后输入参数'bye'
hello too            # 换行后继续输入参数'hello too'，至此接收两个参数
good bye hello too   # 执行"echo good bye hello too"，输出"good bye hello too"
# 通过ctrl+D结束输入
$
```

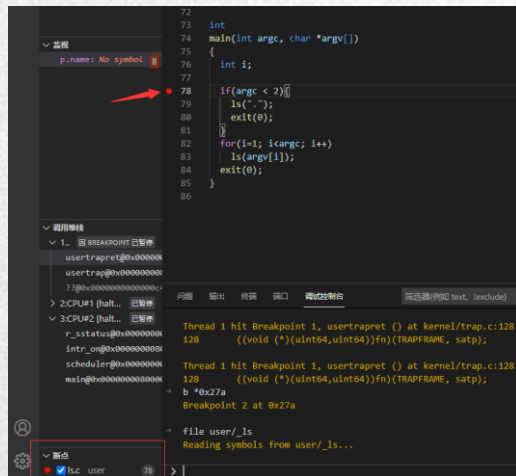
调试技巧

内核态调试 (默认启用)



用户态调试

xv6的内核态和用户态并不共享页表，
需要加载用户程序调试符号(user/_ls)



● 实验提交

• 提交内容

- ① 所有修改过的代码（不需要提交整个xv6文件包）
- ② 实验设计报告

• 截止时间

下一次实验课前提交至HITsz Grader 作业提交平台，具体截止日期参考平台发布。

- 登录网址：：<http://grader.tery.top:8000/#/login>
- 推荐浏览器：Chrome
- 初始用户名、密码均为学号，登录后请修改



THANKS

同学们，
请开始实验吧！