实验一：xv6与Unix实用程序

《操作系统》课程实验

2021秋季

目录

课程安排

实验总分：30分，总共6次实验课

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	懒页分配	简单文件系统的设计与实现
上课时间	第5/6/7周	第8周	第9周	第10周	第12/14周
现场考核	第一次			第二次	第14周第三次

课程考核

- **现场考核/验收(30%，第一次课5%、第4次课10%、最后一次课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%，下次实验课前提前一天提交)**
  - 实验一到实验四提供运行实验设计报告；实验五提交实验报告；
  - xv6相关的实验项目，仅需提交所有修改过的代码；实验五需打包全部工程文件提交。

课程平台

- **课程主页及指导书**地址：https://hitsz-lab.gitee.io/os-labs-2021/index.html
- **演示视频** 下载地址：https://gitee.com/hitsz-lab/os-labs-2021/tree/master/video
- **实验工具** 下载地址（校内网）：http://10.249.11.141/
- **实验提交** 地址（校内网）：http://10.249.12.98:8000/#/login
- **远程实验环境**（校内网）：
  - 计算节点0：10.249.11.143
  - 计算节点1：10.249.11.142

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）：

①MIT OS 6.S081/2020 Lab 1~10的要求：

a) 提交能通过Lab 1~10 自测的完整项目代码；

b) 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；

c) 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；

d) 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！！！

实验目的

- 认识xv6操作系统，初步了解它的背景和特点。
- 学会部署xv6运行的环境，了解xv6源码的结构，编译并运行xv6，从而为整个操作系统实验课程打下基础。
- 巩固理论课程中学习到的系统调用概念，熟悉并学会运用xv6相关的系统调用接口。

实验任务　Part I： 认识xv6

简介

- xv6是由麻省理工学院(MIT)开发以教学目的的操作系统。
- xv6是在x86处理器上用ANSI标准C重新实现的Unix第六版(即v6)，课程编号为6.828。
- 2019年被移植到RISC-V之上，并设置了6.S081课程。

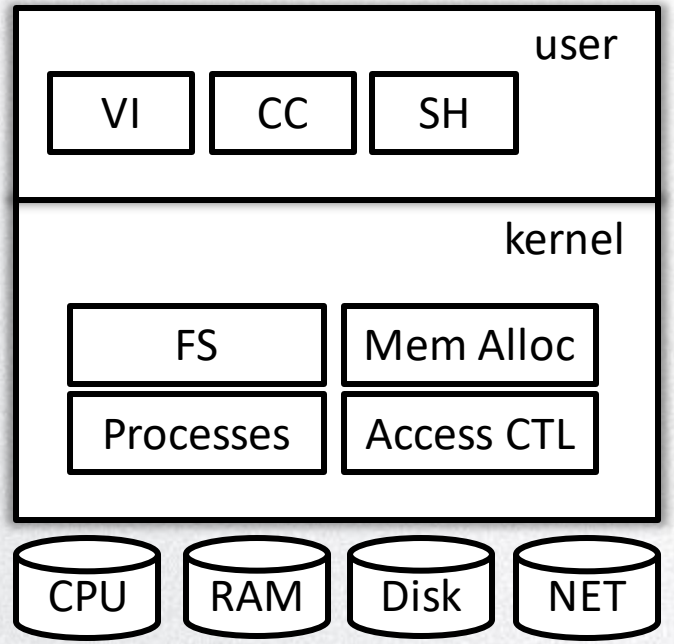
特点

- 功能完善，可理解操作系统原理及实现
- 开源，精简，代码量仅两万行左右
- 基于RISC-V、X86，可深入理解计算机体系结构
- 类Unix系统，可延伸学习其他常见操作系统

实验任务 | Part I： 认识xv6

◆ XV6操作系统结构

- **硬件资源：**CPU、内存、磁盘、网卡
- **user：**vi、cc、sh等各种应用程序
- **Kernel**：计算机资源的守护者
  - 文件系统
  - 进程管理
  - 内存管理
  - 访问控制



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。

实验平台部署方式

- **可直接用的实验环境**
  1. 实验室电脑
  2. 远程实验环境
- **自行部署的实验环境**
  3. 提供VirtualBox + openeuler的镜像，直接导入镜像即可使用
  4. 需自行下载并安装所有工具链，自行编译

实验步骤 | Part II: 编译并运行xv6

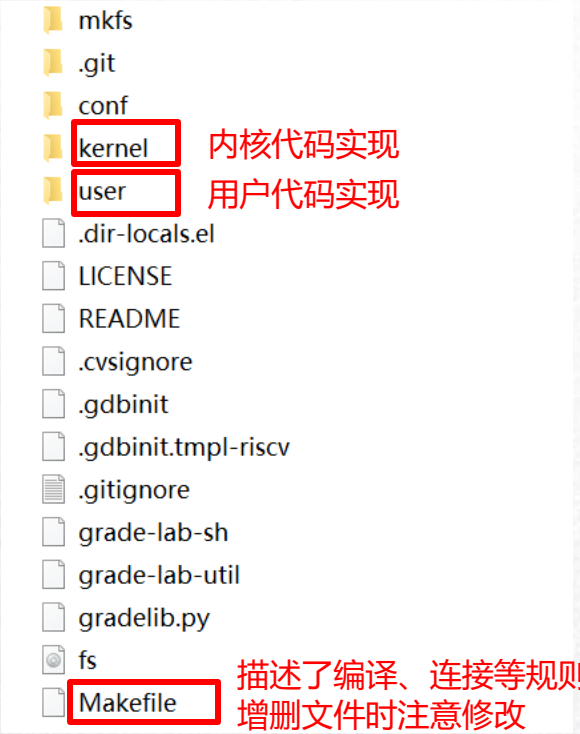
- 请clone最新代码到本地进行实验：

  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:

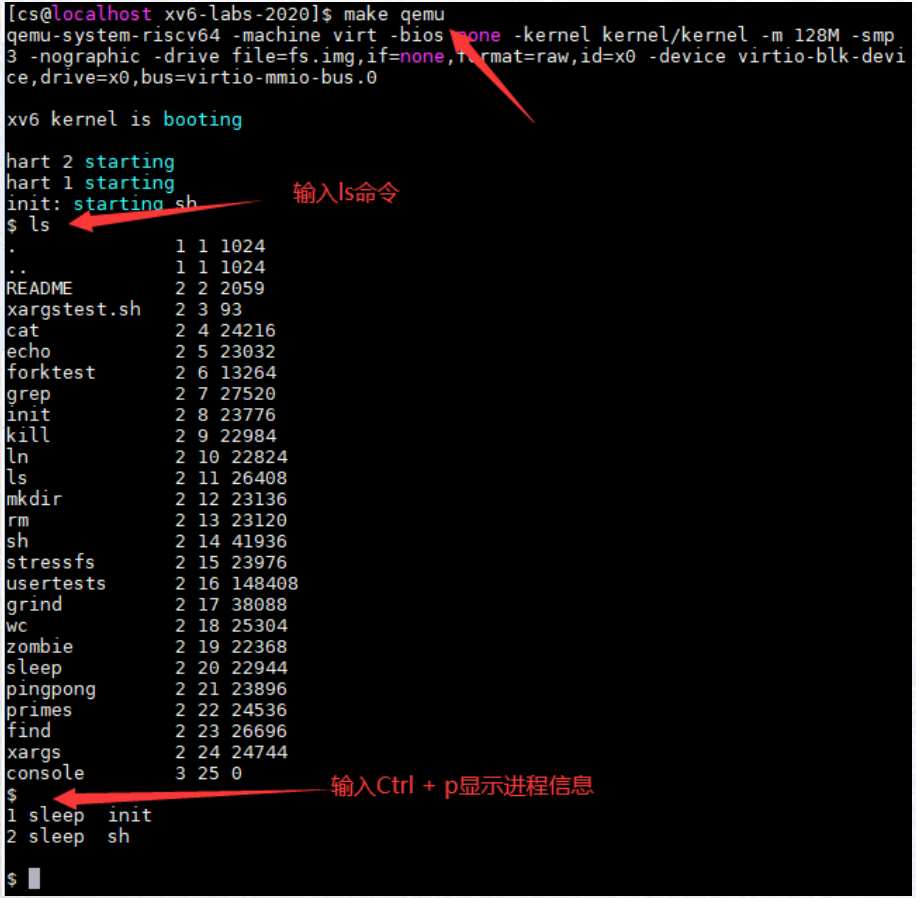
```
$ cd xv6-labs-2020
$ git checkout util
$ git branch
```

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



实验任务 | Part II: 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



实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

◆ 任务一：为xv6实现UNIX程序sleep（使用系统调用sleep使程序睡眠指定滴答数）

- sleep程序会等待用户指定的时间。请将代码写在user/sleep.c文件中。输入命令后，命令行会“暂停”一段时间(10个ticks，ticks由内核定义)，然后输出”(nothing happens for a little while)“。

```
$ make qemu
...
init: starting sh
$ sleep 10
(nothing happens for a little while)
$
```

- 在xv6-labs-2020中，执行下面指令，测试程序：

```
[cs@localhost xv6-labs-2020]$ ./grade-lab-util sleep
make: 'kernel/kernel' is up to date.
== Test sleep, no arguments == sleep, no arguments: OK (1.3s)
== Test sleep, returns == sleep, returns: OK (0.8s)
== Test sleep, makes syscall == sleep, makes syscall: OK (1.0s)
[cs@localhost xv6-labs-2020]$
```

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

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

int main(int argc,char* argv[]){
    if(argc != 2){
        printf("Sleep needs one argument!\n"); //检查参数数量是否正确
        exit(-1);
    }
    int ticks = atoi(argv[1]); //将字符串参数转为整数
    sleep(ticks);              //使用系统调用sleep
    printf("(nothing happens for a little while)\n");
    exit(0); //确保进程退出
}
```

第4行 *int main(int argc, char* argv[]);*

- argc就表示参数的数量，argv则为存储参数所在地址的数组。
- argv的最后一个元素必须为空指针（即数值0），用以标志参数的结束。argv的第一个元素则为需要执行的程序。
- 在shell中执行

$ sleep 10

argc=2，argv=[“sleep", “10", null]

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

```
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 }
```

第5~8行 检查参数

第9行 atoi将字符串转为整数

第10行 使用系统调用sleep

第12行 确保进程退出

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

- 由于sleep.c为新增的用户程序文件，请如下图修改Makefile文件；
- 编译xv6并运行sleep。

```
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/_wc\
	$U/_zombie\
	$U/_cow\
	$U/_uthread\
	$U/_call\
	$U/_kalloctest\
	$U/_bcachetest\
	$U/_mounttest\
	$U/_crashtest\
	$U/_sleep\
```

```
xv6 kernel is booting

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

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

◆ 任务二：在xv6上实现pingpong程序

即两个进程在管道两侧来回通信。父进程将“ping”写入管道，子进程从管道将其读出并打印。子进程从父进程收到字符串后，将“pong”写入另一个管道，然后由父进程从该管道读取并打印。请将代码写在user/pingpong.c文件中。运行效果应该如下：

```
$ make qemu
...
init: starting sh
$ pingpong
4: received ping
3: received pong
$
```

➢ 在xv6-labs-2020中，执行下面指令，测试程序：

```
[cs@localhost xv6-labs-2020]$ ./grade-lab-util pingpong
make: 'kernel/kernel' is up to date.
== Test pingpong == pingpong: OK (1.1s)
[cs@localhost xv6-labs-2020]$
```

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

◆ **提示:**

- 使用系统调用pipe创建管道。
- 使用系统调用fork创建子进程。
- 使用系统调用read从管道中读取数据，使用系统调用write向管道中写入数据。
- 使用系统调用getpid获取调用进程的进程ID
- 将程序添加到Makefile中的UPROGS。

- xv6可用库函数
  - user/user.h内容如下

```
// system calls
int fork(void);
int exit(int) __attribute__((noreturn));
int wait(int*);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);

// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
void fprintf(int, const char*, ...);
void printf(const char*, ...);
char* gets(char*, int max);
uint strlen(const char*);
void* memset(void*, int, uint);
void* malloc(uint);
void free(void*);
int atoi(const char*);
int memcmp(const void *, const void *, uint);
void *memcpy(void *, const void *, uint);
```

系统调用

库函数

user/ulib.c
user/printf.c
user/umalloc.c

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

◆ 任务三：在xv6上使用管道实现“质数筛选”

使用管道编写素数筛法（prime sieve）的并发版本。这个想法是由Unix管道的发明者Doug McIlroy提出的。请编写程序输出2~35之间的所有质数，在user/primes.c实现。运行效果应该如下：

```
$ make qemu
...
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
$
```

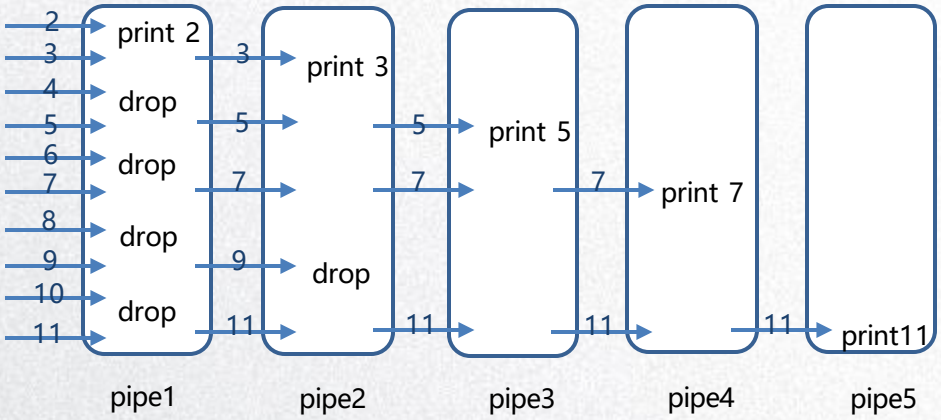
➢ 在xv6-labs-2020中，执行下面指令，测试程序：

```
[cs@localhost xv6-labs-2020]$ ./grade-lab-util primes
make: 'kernel/kernel' is up to date.
== Test primes == primes: OK (0.9s)
[cs@localhost xv6-labs-2020]$
```

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

◆ 提示：

- 由于xv6的文件描述符和进程数量有限，请注意关闭进程不需要的文件描述符，否则，你的程序在xv6上运行会在第一个进程到达35之前耗完资源。
- 一旦第一个进程到达35，它应该等到整个管道终止，包括所有子进程、孙进程等等。因此，primes的主进程应该在输出完所有输出和所有其他子进程退出之后退出。



2是素数，将2的整数倍的数去掉，
3是素数，将3的整数倍的数去掉，依次类推。

错误示例

```
$ primes
prime 2
prime 3
prime 5
prime 7
prime 11
prime 13
prime 17
prime 19
prime 2$ 3
prime 29
prime 31
```

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

◆ 任务四：在xv6上实现用户程序find

在目录树中查找名称与字符串匹配的所有文件，输出文件的相对路径。该程序的命令格式为“find path file_name”。请将代码写在user/find.c文件中。运行效果应该如下：

```
$ make qemu
...
init: starting sh
$ echo > b
$ mkdir a
$ echo > a/b
$ find . b
./b
./a/b
$
```

➢ 在xv6-labs-2020中，执行下面指令，测试程序：

```
[cs@localhost xv6-labs-2020]$ ./grade-lab-util find
make: 'kernel/kernel' is up to date.
== Test find, in current directory == find, in current directory: OK (1.5s)
== Test find, recursive == find, recursive: OK (2.0s)
[cs@localhost xv6-labs-2020]$
```

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

◆ 简单版的Unix find程序

◆ 提示：

- 查看user/ls.c以了解如何读取目录。
- 使用递归允许find可进入子目录查找。
- 不要递归到“.”和“..”。
- 在qemu运行期间文件系统会被持续更改，要获得干净的文件系统，请先运行make clean，然后运行make qemu。
- 需要使用C字符串。注意==不像Python那样比较字符串。请用strcmp()。
- 将程序添加到Makefile文件的UPROGS中。

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

◆ 任务五：在xv6上实现用户程序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给该命令。请将代码写在user/xargs.c中。运行效果应该如下：

```
$ make qemu
...
init: starting sh
$ sh < xargstest.sh
$ $ $ $ $ $ hello
hello
hello
$ $
```

```
$ make qemu
...
$ echo 3 4|xargs echo 1 2
1 2 3 4
$
```

➢ 在xv6-labs-2020中，执行下面指令，测试程序：

```
[cs@localhost xv6-labs-2020]$ ./grade-lab-util xargs
make: 'kernel/kernel' is up to date.
== Test xargs == xargs: OK (2.9s)
[cs@localhost xv6-labs-2020]$
```

实验任务 | Part III: 利用xv6系统调用实现5个Unix实用程序

◆ 提示：

- 在每一行输入上使用fork和exec调用该命令。在父进程中使用wait等待子进程完成命令。
- exec接收的二维参数数组argv，argv[0]必须是该命令本身，最后一个参数argv[size-1]必须为0，否则将执行失败。

```
char *argv[] = { "echo", "hello", 0 };
exec("echo", argv);
```

- 要读取单独的输入行，每次读取一个字符，直到出现换行符('\n')。
- 可以使用kernel/param.h中定义的MAXARG来声明argv的长度；
- 将程序添加到Makefile的UPROGS中。
- 在qemu运行期间文件系统会被持续更改，请运行make clean，然后运行make qemu。

实验要求

- 4个程序（sleep已给出源码）都能正常运行（分数权重均等），可以通过如下测试：

```
xv6@debian:~/Desktop/share/xv6-riscv-fall19$ ./grade-lab-util
make: 'kernel/kernel' is up to date.
sleep, no arguments: OK (1.5s)
sleep, returns: OK (1.2s)
sleep, makes syscall: OK (1.2s)
pingpong: OK (1.2s)
primes: OK (1.3s)
find, in current directory: OK (1.9s)
find, recursive: OK (2.5s)
xargs: OK (2.6s)
Score: 100/100
```

实验提交

- 第二次实验课前一天，提交所有修改过的代码与实验设计报告

  压缩文件命名为：学号_姓名_OSLab1

  http://10.249.12.98:8000/#/login

实验报告下载链接：

http://10.249.11.141/oslab/

THANKS

同学们，
请开始实验吧！