

Lab1 UTIL: XV6与Unix应用程序

操作系统实验指导书 - 2024秋季 [实验概述 \(https://elearning.fudan.edu.cn/courses/78523/pages/shi-yan-gai-shu\)](https://elearning.fudan.edu.cn/courses/78523/pages/shi-yan-gai-shu) | [常见问题汇总 \(https://elearning.fudan.edu.cn/courses/78523/pages/chang-jian-wen-ti-hui-zong\)](https://elearning.fudan.edu.cn/courses/78523/pages/chang-jian-wen-ti-hui-zong) | **Lab1 UTIL** | [Lab2 SYSCALL \(https://elearning.fudan.edu.cn/courses/78523/pages/lab2-syscall-xi-tong-diao-yong\)](https://elearning.fudan.edu.cn/courses/78523/pages/lab2-syscall-xi-tong-diao-yong) | [Lab3 Scheduling \(https://elearning.fudan.edu.cn/courses/78523/pages/lab3-scheduling-jin-cheng-diao-du\)](https://elearning.fudan.edu.cn/courses/78523/pages/lab3-scheduling-jin-cheng-diao-du) | [Lab4 Page table \(https://elearning.fudan.edu.cn/courses/78523/pages/lab4-page-table-ye-biao\)](https://elearning.fudan.edu.cn/courses/78523/pages/lab4-page-table)

一、实验目的

- 认识XV6操作系统，并熟悉其运行环境。
- 复习并巩固系统调用、进程等理论知识，掌握在XV6上编写用户程序的方法，要求实现 `sleep`、`pingpong` 和 `find` 用户程序。
- 理解和掌握XV6的启动流程，包括资源初始化、第一个进程的诞生等，要求在启动流程涉及到的 `start`、`main` 等函数中增加打印输出你的学号以及该函数的作用。

二、实验准备

2.1 部署实验环境

参考[实验概述 \(https://elearning.fudan.edu.cn/courses/78523/pages/shi-yan-gai-shu\)](https://elearning.fudan.edu.cn/courses/78523/pages/shi-yan-gai-shu)中部署XV6环境相关资料，配置XV6运行环境，并将实验用代码从代码仓库中clone到本地。

重要

注意如果已经下了之前Gitee仓库的麻烦重新拉取xv6代码仓库，谢谢！

```
$ git clone https://github.com/Chalkydoge/xv6-oslab24.git
```

本课程实践中，每次Lab都在不同的分支上完成，请注意切换分支。Lab1 UTIL需要切换到util分支后进行开发。

```
$ git branch -a  
$ git checkout util
```

2.2 熟悉Unix实用程序

本次实验需要编写实验内容中介绍的3个Unix实用程序。初次接触操作系统实验的你可能会感到不知所措，因此不妨先体验一下这些程序的运行效果。实际上，Linux中具备本次实验要实现的一些程序，例

如 `sleep`、`find`。你可以先尝试在Linux中使用这些命令，充分体会功能后再开始编程。当然，Linux中命令的功能较为复杂，我们仅要求实现简化版。

实验开始之前，我们**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`。

三、实验内容

3.1 sleep

- 实验目标

了解XV6上用户程序 `sleep` 的实现。`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。

- 预期结果

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

输入命令后，命令行会"暂停"一段时间 (10个ticks，ticks由内核定义，一个tick大概100ms)，然后输出"(nothing happens for a little while)"。

- 本地测试

在 `xv6-oslab24-hitsz` 中，执行下面指令 `python3 grade-lab-util sleep`，测试程序：

或者 `ln -s /usr/bin/python3 /usr/bin/python` 然后用 `./grade-lab-util [你需要测试的程序名字]`

```
test_19@compl:~/xv6-oslab23-hitsz$ ./grade-lab-util sleep
make: 'kernel/kernel' is up to date.
== Test sleep, no arguments == sleep, no arguments: OK (1.4s)
== Test sleep, returns == sleep, returns: OK (0.9s)
== Test sleep, makes syscall == sleep, makes syscall: OK (0.9s)
test_19@compl:~/xv6-oslab23-hitsz$
```

- 提示

在Makefile中加入编译 `sleep.c` 的任务，在合适的地方插入如下语句：

```
$U/_sleep\
```

3.2 pingpong

- 实验目标

在XV6上实现 `pingpong` 程序，即两个进程在管道两侧来回通信。请将代码写在 `user/pingpong.c` 文件中。

- 预期结果

父进程向管道中写入数据，子进程从管道将其读出并打印 `<pid>: received ping from pid <father pid>`，其中 `<pid>` 是子进程的进程ID，`<father pid>` 是父进程的进程ID。子进程从父进程收到数据后，通过写入另一个管道向父进程传输数据，然后由父进程从该管道读取并打印 `<pid>: received pong from pid <child pid>`，其中 `<pid>` 是父进程的进程ID，`<child pid>` 是子进程的进程ID。

```
root@d119nb:/home/pei/xv6-oslab24-hitsz# ls
Dockerfile LICENSE Makefile README __pycache__ clang-format.py conf fs.img grade-lab-util gradelib.py kernel mkfs user xv6.out xv6.out.pingpong
root@d119nb:/home/pei/xv6-oslab24-hitsz# python3 grade-lab-util pingpong
make: 'kernel/kernel' is up to date.
== Test pingpong lenient testing == pingpong lenient testing: OK (0.7s)
(Old xv6.out.pingpong failure log removed)
== Test pingpong strict testing with changing pids == pingpong strict testing with changing pids: OK (1.0s)
root@d119nb:/home/pei/xv6-oslab24-hitsz#
```

- 本地测试

在 `xv6-oslab24` 中（注意切换本地git仓库 `branch=util`），执行下面指令测试程序：

```
$ grade-lab-util pingpong
```

- 提示

1. 你需要使用两个管道(可以命名为 `c2f` 和 `f2c`)；
2. 其中 `c2f` 用于子进程向父进程传输数据；
3. `f2c` 用于父进程向子进程传输数据。（思考一下能不能只用一个管道实现呢？答案是并不能）

3.3 find

- 实验目标

在XV6上实现用户程序 `find`，即在目录树中查找名称与字符串匹配的所有文件或目录，输出文件的相对路径。该程序的命令格式为 `find <path> <name>`。请将代码写在 `user/find.c` 文件中。

- 预期结果

在某目录中新建 文件**b**或 目录**b**，在当前目录下查询 `b` 的输出效果应该如下：

```
init: starting sh
$ echo > b
$ find . b
./b
$

init: starting sh
$ mkdir b
$ find . b
./b
$
```

另外还应当实现递归查询，当要查询的文件 `target` 在根目录的某个子文件夹中时，应该能将结果显示出来：

```
init: starting sh
$ mkdir a
$ echo > a/target
$ mkdir a/b
$ mkdir a/b/target
$ mkdir c
$ echo > c/target
$ find . target
./a/target
./a/b/target
./c/target
$
```

- 本地测试

在 `xv6-oslab24` 中，执行下面指令 `python3 grade-lab-util find`，测试程序：

```
> ./grade-lab-util find
make: 'kernel/kernel' is up to date.
== Test find, in current directory and create a file == find, in current directory and create a file: OK (1.8s)
== Test find, in current directory and create a dir == find, in current directory and create a dir: OK (0.8s)
== Test find, find file recursive == find, find file recursive: OK (1.3s)
== Test find, find dir recursive with no duplicates == find, find dir recursive with no duplicates: OK (1.0s)
```

如果上述3个程序(`sleep`、`pingpong`、`find`)都能正常运行，可以到 `xv6-oslab24-hitsz` 项目目录，执行 `make grade` 指令测试：

```
make[1]: Leaving directory '/root/other0s/xv6-oslab24-hitsz'
== Test sleep, no arguments ==
$ make qemu-gdb
sleep, no arguments: OK (2.6s)
== Test sleep, returns ==
$ make qemu-gdb
sleep, returns: OK (1.2s)
== Test sleep, makes syscall ==
$ make qemu-gdb
sleep, makes syscall: OK (1.0s)
== Test pingpong lenient testing ==
$ make qemu-gdb
pingpong lenient testing: OK (1.1s)
== Test pingpong strict testing with changing pids ==
$ make qemu-gdb
pingpong strict testing with changing pids: OK (0.8s)
== Test find, in current directory and create a file ==
$ make qemu-gdb
find, in current directory and create a file: OK (1.5s)
== Test find, in current directory and create a dir ==
$ make qemu-gdb
find, in current directory and create a dir: OK (0.9s)
== Test find, find file recursive ==
$ make qemu-gdb
find, find file recursive: OK (1.3s)
== Test find, find dir recursive with no duplicates ==
$ make qemu-gdb
find, find dir recursive with no duplicates: OK (1.2s)
Score: 60/60
```

3.4 xv6启动流程实验

- 实验目标

学会使用GDB调试XV6启动流程（请跳过GDB-dashboard直接使用GDB进行调试，GDB打印出 `initcode` 和 `init` 的调试指令即可）。请将代码写在 `user/commands.gdb` 文件中。

- 预期结果

在调试过程中需要在两处地方使用 `p cpus[$tp]->proc->name`（`$tp` 是TP寄存器的值，XV6使用它作为CPU的编号，这串式子其实相当于 `myproc()` 函数）打印出当前 `proc` 结构体的 `name`，分别打印出初始进程 `initcode` 和 `init` 程序的名字。要求在GDB调试中使用 `source commands.gdb` 执行脚本后，会显示如下内容（下面演示了 `initcode` 的调试流程）

```

Breakpoint 2, scheduler () at kernel/proc.c:415
415 void scheduler(void) {
(gdb) n
417 struct cpu *c = mycpu();
(gdb) n
419 c->proc = 0;
(gdb) n
422 intr_on();
(gdb) n
425 for (p = proc; p < &proc[NPROC]; p++) {
(gdb) n
426 acquire(&p->lock);
(gdb) n
427 if (p->state == RUNNABLE) {
(gdb) n
431 p->state = RUNNING;
(gdb) n
432 c->proc = p;
(gdb) n
433 swtch(&c->context, &p->context);
(gdb) p cpus[$tp]->proc->name
$1 = "initcode\000\000\000\000\000\000\000"
(gdb)

```

调试方法

打开终端，到 xv6-oslab24 目录下输入：

```
make qemu-gdb CPUS=1
```

然后再新开一个终端，在相同目录下输入

```
make gdb
```

进入这个界面，即可进行正常调试

- 提示

最终的 `commands.gdb` 文件（不限制长度，长度不算在分数内，但最短仅需6行，同学们可以尝试）应该类似这样

```

...
p cpus[$tp]->proc->name # 应该先打印出 "initcode"
...
p cpus[$tp]->proc->name # 再打印出 "init"
da # 使用命令刷新 dashboard 并将上一个输出显示在 history 区域

```

提交：请将调试过程使用的GDB指令写入 `commands.gdb` 文件，具体可以copy根目录下 `.gdb_history` 中的内容（这个文件内会存储使用gdb调试的历史记录）

3.5 问答题

了解管道模型，回答下列问题：

1. 简要说明在pingpong实验中，你是怎么创建管道的？结合fork系统调用说明你是怎么使用管道在父子进程之间传输数据的。
2. 试解释，为什么要提前关闭管道中不使用的一端？（提示：结合管道的阻塞机制）
3. 阅读如下示例程序。

```
int p[2];          /* 存储管道的两个文件描述符 */
char *argv[2];
argv[0] = "wc";    /* 设置要执行的命令为"wc" */
argv[1] = 0;       /* 参数数组以NULL结尾，表示没有更多参数 */
pipe(p);           /* 创建管道，p[0]是管道的读端，p[1]是管道的写端 */

if (fork() == 0) {
    /* 子进程 */
    close(0);       /* 关闭标准输入（文件描述符0） */
    dup(p[0]);      /* 复制管道的读端p[0]，让文件描述符0指向管道的读端 */
    close(p[0]);    /* 关闭重复的管道读端 */
    close(p[1]);    /* 关闭不再需要的管道写端 */
    exec("/bin/wc", argv); /* 执行"wc"程序 */
} else {
    /* 父进程 */
    close(p[0]);    /* 关闭管道的读端 */
    write(p[1], "hello world\n", 12); /* 向管道写入"hello world\n" */
    close(p[1]);    /* 关闭管道的写端，表示写入完成 */
}
```

假设子进程没有关闭管道写端，运行该程序后，尝试描述会发生什么？为什么会有这样的结果？

请在实验报告中回答以上问题。

四、实验结果提交

将实验报告和实验代码打包为压缩文件提交到eLearning平台。

4.1 实验报告

参照[课程文件](#)

(<https://elearning.fudan.edu.cn/courses/78523/files/folder/%E5%AE%9E%E9%AA%8C%E7%9B%B8%E5%85%B3%E6%96%87%E6%A1%A3>)中的《OS实验报告模板》


(<https://elearning.fudan.edu.cn/files/4915949/>)，按如下要求书写实验报告，力争规范、简洁。

1. 对于每个实验，详细描述实验过程，对于你认为的关键步骤附上必要的截图。
2. 有需要写代码的实验，必须配有代码、注释以及对代码功能的说明。
3. 鼓励实验报告中包括但不局限于以下内容：实验过程中碰到了什么问题？如何解决这些问题？实验后还存在哪些疑问或者有什么感想？

4. 如果实验附有练习，请在每个练习之后作答，这是实验报告评分的重要部分。

4.2 实验代码

不需要提交完整的代码包，只需要提交 `commit.patch` 文件即可，操作步骤如下：

- 在完成实验之后，将当前分支上的所有更改进行提交（`commit`，具体方法参考[git使用教程](https://os-labs.pages.dev/lab1/part4/#3-git)  (<https://os-labs.pages.dev/lab1/part4/#3-git>)。
- 在仓库的目录下使用 `make diff` 命令导出更改文件 `commit.patch`。

4.3 提交平台

请将生成的 `commit.patch` 文件与实验报告一起打包提交到eLearning平台[Lab1 UTIL: XV6与Linux应用程序](https://elearning.fudan.edu.cn/courses/78523/assignments/95742) (<https://elearning.fudan.edu.cn/courses/78523/assignments/95742>)。

操作系统实验指导书 - 2024秋季 [实验概述](https://elearning.fudan.edu.cn/courses/78523/pages/shi-yan-gai-shu) (<https://elearning.fudan.edu.cn/courses/78523/pages/shi-yan-gai-shu>) | [常见问题汇总](https://elearning.fudan.edu.cn/courses/78523/pages/chang-jian-wen-ti-hui-zong) (<https://elearning.fudan.edu.cn/courses/78523/pages/chang-jian-wen-ti-hui-zong>) | [Lab1 UTIL](#) | [Lab2 SYSCALL](https://elearning.fudan.edu.cn/courses/78523/pages/lab2-syscall-xi-tong-diao-yong) (<https://elearning.fudan.edu.cn/courses/78523/pages/lab2-syscall-xi-tong-diao-yong>) | [Lab3 Scheduling](#) (<https://elearning.fudan.edu.cn/courses/78523/pages/lab3-scheduling-jin-cheng-diao-du>) | [Lab4 Page table](https://elearning.fudan.edu.cn/courses/78523/pages/lab4-page-table-ye-biao) (<https://elearning.fudan.edu.cn/courses/78523/pages/lab4-page-table-ye-biao>)