Lab1：实验平台搭建和几个练手程序

1.1 预备知识

本课程的实验平台是 xv6 操作系统内核，大家可以先去它的主页观察一下有关 xv6 的前生今世。https://pdos.csail.mit.edu/6.828/2022/xv6.html

然后，xv6 的原始资料有两个：第一个是 xv6 的“官方参考书”，它从各个方面对 xv6 进行了代码级别的介绍：

https://pdos.csail.mit.edu/6.828/2023/xv6/book-riscv-rev3.pdf

第二个则是 xv6 的“官方实验指南”，指的是对每一次实验本身做的一些任务介绍，它存在于 https://pdos.csail.mit.edu/6.828/2023/schedule.html 网页上。其中，像带 Lab 字样的文本超链接，比如“Lab util：UNIX utilities”和“Lab syscall：System calls”这样的，就指的是需要自己动手编程的实验。

官方实验指南给出的描述过程比较浓缩，不是太好懂，因此，除了你自己每次实验前以它为核心多读几遍外，这里我自己也补充第三个资料，就是本文档本身，暂时称之为“中文实验文档”。

大家要注意，虽然中文实验文档会是官方实验指南的一个补充，但是也不会把实验的每一个细节说的清清楚楚，这个时候，大家就要发挥自己的主观能动性：多看、多查、多问、多动手、多思考。

现阶段，xv6 的主要用途是用于教学，主要的编程语言是 C 和少量汇编语言。在汇编语言这一块，对应支持 xv6 的是两类指令集：一个是大家可能熟悉的 X86 指令集（英特尔和 AMD）；另一个是最近兴起的开源指令集 RISC-V。关于这两者之间，还有与如日中天的 ARM 指令集之间的对比，我在下面贴一段话给大家做理解和参考，感兴趣的同学可以深究。出于开放性和学术性的考虑，这里选择 xv6 之 RISC-V 实现作为本课程的实验内核。

> RISC-V is a rather old instruction set by today's standard but has only recently received wide adoption by the computer industry. RISC-V has many advantages over mainstream instruction sets like x86, AMD64, and ARM. The largest of them is full open-source licensing so anyone can utilize the architecture. It is radically different from x86 and ARM, which require commercial licenses and fees for companies to acquire and use the technology.
>
> From a performance standpoint, the most differentiating feature of RISC-V compared to x86 and AMD64 is the use of a more simplistic RISC architecture instead of CISC.

> RISC or Reduced Instruction Set Computer follows the idea of using simple instructions completed in a single clock cycle. The disadvantage of this technique is that it requires more code to complete an executed task, but the advantages are improved battery life and energy efficiency. (If you were wondering, ARM chips also use RISC over CISC to improve battery life on mobile devices, but its implementation is far different.)
>
> CISC, or Complex Instruction Set Computer, is the opposite of RISC. Instead of completing instructions in a single clock cycle, the goal is to complete tasks in as few lines of code as possible, which frequently means instructions will take multiple clock cycles to complete. It results in higher energy consumption, but it is more advantageous to developers since CISC requires less code altogether.
>
> RISC-V is perfectly capable of being used in the consumer space; however, competition from other instruction sets has relegated it to the enterprise markets. Nearly all chips built on RISC-V cater to HPC, AI, or other high-level computing tasks, but RISC-V can break out of the mold if more laptops and desktops come out featuring these types of chips.

1.2 实验平台的搭建

用心的同学看完 1.1 的内容，可能会发出“我只有基于 ARM 的 Apple Max Pro 的 CPU，根本没有 RISC-V 的 CPU，怎么去运行这个 xv6！”类似的怒吼。其实，不管你用的是 Apple 的 M1、M2 或者 M3 芯片，还是基于传统 X86 的英特尔和 AMD 的芯片，都可以通过硬件级别的系统虚拟化技术（也是云计算的核心技术之一），来运行基于 RISC-V 的 xv6。在这里，我们要用到一个非常著名的开源虚拟化软件 QEMU（https://www.qemu.org/）。它可以通过“动态二进制翻译”的方法来把 RISC-V 的指令集转换成“等价”的 ARM 指令集或者 X86 指令集。

因此，如果我们要在自己的机器上安装实验环境的话，你的实验平台的体系结构应该是如下图所示：

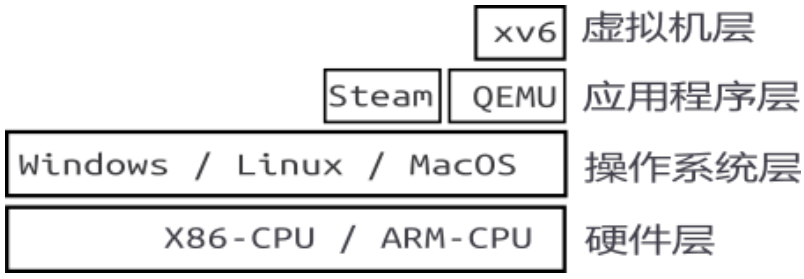

图 1.1 《操作系统课程项目》的实验平台体系结构：用 QEMU 来进行从 RISC-V 指令集到 X86 或者 ARM 指令集之间的转换，并保持主机系统（即图中操作系统层）与实验环境（即图中的虚拟机层）的隔离性。

在“硬件层”，我们不需要专门的额外硬件或机器，就用各位现有的台式机或者笔记本电脑就足以应付。

“操作系统层”，又叫做“主机系统层”。相信大家现有的主机系统不会超过

Windows/Linux/MacOS/FreeBSD 的范围，所以也不需要折腾什么，QEMU 也针对各类平台都有对应的版本。但是，因为各位还处于本科专业学习阶段，我个人建议把 Linux 作为自己实验平台的主机系统（即图 1.1 中的操作系统层/主机系统层）。看到这里，有的同学会比较困惑：因为自己只有一台机器，并且已经装了像 Windows 或者 MacOS 这样的商用主机系统（确实也好用），如果为了一门 1 个学分的实验课而推翻原有的学习和研发环境（娱乐环境？），是否会对其他正在进行的课程或任务造成负面影响？

其实，我们可以继续采用系统虚拟化技术来进行折衷权衡：在图 1.1 中的操作系统层使用像 VMware Workstation、Oracle VirtualBox 和 Parallel Desktop 这样的虚拟机软件（也是应用程序）来安装一个 Linux 发行版，比如 Ubuntu Server（20+版本），然后在这个 Ubuntu Server 的内部，再去安装对应版本的 QEMU，最后，借助于这个虚拟机内部的 QEMU，去运行 xv6-riscv。所以，我们原本的实验平台体系结构，就会变成下面的，略微“复杂”和“奇葩”的图 1.2：

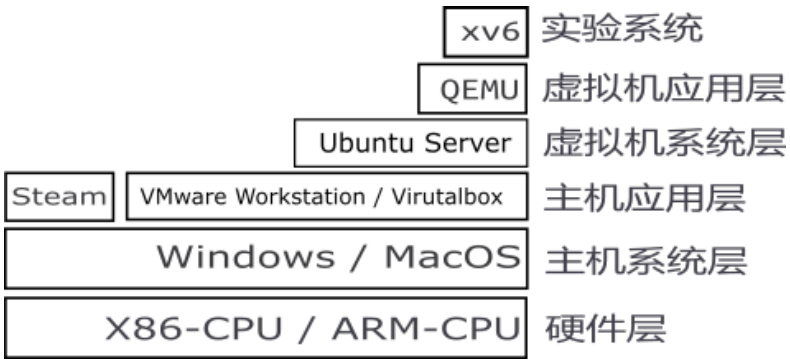

图 1.2“折衷权衡”后的实验平台体系结构：可以看到，整个系统中出现了两个在层次上完全不一样的虚拟机：Windows/MacOS 上运行了 Ubuntu Server 虚拟机（通过 VMware 或者 Virtualbox 模拟），而 Ubuntu Server 内部，又有一个 xv6 的虚拟机（通过 QEMU 模拟）。

> 这种在主机系统里运行一个虚拟机系统，然后在后者内部再运行一个虚拟机系统的软件结构，被称为“嵌套”虚拟化（有没有“盗梦空间”的内味？），而这种技术可以在诸如虚拟机系统调试、系统模拟和恶意软件隔离和检测等场合发挥作用。在 2010 年的操作系统领域最顶尖的学术会议之一——OSDI（USENIX Symposium on Operating Systems Design and Implementation）上，一篇最佳学术论文的主要工作就是对嵌套虚拟机系统进行性能 I/O 优化（点此链接）。

当然，上面这种看起来头晕的嵌套式虚拟机实验环境搭建方式只是可选的（我在自己的 Windows 11 Pro 的移动工作站里就是用到的这种搭建方式），你完全可以在自己的主机系统里直接安装对应版本的 QEMU（*因为少数同学反应这种方式会导致一些错误，本人还是建议各位使用图 1.2 的方式或者主机系统直接安装 ubuntu server 来运行 xv6-riscv*）。下面两个

链接分别有告诉大家怎么在自己的主机系统或者虚拟机系统里安装 QEMU 和构建 xv6-riscv,:

- https://pdos.csail.mit.edu/6.828/2021/tools.html
- https://pdos.csail.mit.edu/6.828/2021/labs/util.html

基本的步骤无非是:

(1) 在主机系统里安装对应版本的 QEMU 及其对应的依赖包，而主机系统最好是完整的 ubuntu server （20+以上版本);

(2) 如果主机系统不是 ubuntu server，可以用图 1.2 的嵌套虚拟化方案;

(3) 直接在 Windows 的 WSL 和 MacOS 安装 qemu（少数同学会出现不可预料的问题);

(4) 用 git 相关命令下载 xv6 的源码至主机系统或者 ubuntu 虚拟机;

(5) 用 git 相关命令切换代码分支（每个实验都有其对应的代码分支);

(6) 编译+运行 xv6。

几个注意的地方:

(1) 这里会用到 git 这个开源的分布式代码版本控制系统，之前大家没学过的可以自己去学一下，不需要太复杂，会几个简单的 git 命令即可；如果自己的主机系统上没有 git 工具，可以在这里安装一个: https://www.git-scm.com/;

(2) git clone xv6 的代码仓库的时候，速度可能会比较慢，请保持耐心;

(3) 我们不用 Athena 的公共主机，大家就在自己机器上折腾即可;

(4) 我们不需要往 MIT 的代码仓库提交作业，也暂时不需要自动评分系统，大家千万不要去往 MIT 的代码仓库的方向去提交相关和不相关的东西（也没这权限)，以免 SCNU 的 IP 被封。

1.3 下载、编译和启动 xv6

如果没有特别说明，下面的术语都以图 1.1 的体系结构为准。

本章节的实验步骤基本按照此链接进行（也即官方实验指南)，但不囊括其全部内容——关于其描述的提交作业的部分可以忽略不看，因为我们目前只需要在完成作业后，在学者网对应的作业提交入口提交实验报告即可。

(A) 根据 1.2 小节提供的信息，我们在主机系统上装好 qemu 和 git 工具后，可以在命令行输入下面的 git clone 命令，把要用到的 xv6-riscv 源码下载至本地（本

地指的是图 1.1 中的操作系统层或者图 1.2 中的虚拟机系统层）：

```
$ git clone git://g.csail.mit.edu/xv6-labs-2021    //我这里以 2021 的版本为例
Cloning into 'xv6-labs-2021'...
```

（B）耐心等待并成功之后，进入 xv6 的源码目录，并切换到名为“util”的代码分支：

```
$ cd xv6-labs-2021
$ git checkout util
Branch 'util' set up to track remote branch 'util' from 'origin'.
Switched to a new branch 'util'
```

（C）编译并进入 xv6 系统：在主机系统对应的 xv6 源码目录下，敲：

```
$ make qemu
```

如果顺利的话，你会看见如下信息，就表示已经进入了 xv6 的交互界面：

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

此时命令行（终端）是在 xv6 的系统内部了。如果你想关闭 xv6，重新回到主机系统的终端，按下 ctrl+a，松掉按键，再按 x 键就退出了（此时 xv6 虚拟机已经关闭了）。

1.4 三个练手小程序

大家先通读几遍本次实验的官方实验文档：

https://pdos.csail.mit.edu/6.828/2021/labs/util.html

我们需要在本次实验中完成其中的三个程序：（1）sleep；（2）pingpong 和（3）primes。剩下的（4）find 和（5）xargs 不强制要求。但是，目前找工作和升学的内卷惨烈程度大家都有所耳闻，如果现在不找机会拼命提高自己的能力，还要等到什么时候？

1.4.1 sleep

这个 sleep 的程序是 xv6 的应用层程序，它的主要运行逻辑是：在终端进入睡眠状态，该状态持续的时间为用户输入的数值参数 t。而 t 的准确定义是：t 个由 xv6 定义的时间单

位，一个时间单位为两次连续时间中断事件之间的延时（中断来自于底层的硬件计时器）。

此程序要用到 xv6 已有的系统调用——int sleep(int n)，但是需要 include 一些对应的头文件或库。大家可以在 xv6 源码目录中找到名为 user/的文件夹，这里已经有一些样本程序供大家参考，例如：user/echo.c，user/grep.c，and user/rm.c。通过这些已有的程序，可以了解如何把 *t* 参数传进 sleep 程序的方法，也可以学习一下简洁清晰的编码写作风格。

假设你编写的程序名为“sleep.c”。在编写完成后：

(1) 请将“sleep.c”拷贝至 xv6 源码目录下的 user/文件夹下（图 1.1 主机系统层或者图 1.2 的虚拟机系统层）；

(2) 然后在 xv6 源码根目录下的 Makefile 文件里，找到关键字“UPROGS=\”（可能是第 179 行左右），在其下面列出的多个以“&U”开头的字符串的下方（可能是第 195 行左右），添加一行“&U/_sleep\”，并保存；

(3) 回到 xv6 的源码目录的根目录下，敲“make qemu”，以让 sleep.c 得以编译并在 xv6 内部形成可执行程序；

(4) 顺利进入到 xv6 的终端后，敲“./sleep 5”；

(5) 如果没有出错信息，该程序会在终端等待几秒钟延迟后，又回到了 xv6 的命令提示符。

其他的提示请看下面：

Some hints:

- Before you start coding, read Chapter 1 of the xv6 book.
- Look at some of the other programs in user/ (e.g., user/echo.c, user/grep.c, and user/rm.c) to see how you can obtain the command-line arguments passed to a program.
- If the user forgets to pass an argument, sleep should print an error message.
- The command-line argument is passed as a string; you can convert it to an integer using atoi (see user/ulib.c).
- Use the system call sleep.
- See kernel/sysproc.c for the xv6 kernel code that implements the sleep system call (look for sys_sleep), user/user.h for the C definition of sleep callable from a user program, and user/usys.S for the assembler code that jumps from user code into the kernel for sleep.
- Make sure main calls exit() in order to exit your program.
- Add your sleep program to UPROGS in Makefile; once you've done that, make qemu will compile your program and you'll be able to run it from the xv6 shell.

- Look at Kernighan and Ritchie's book The C programming language (second edition) (K&R) to learn about C.

1.4.2 pingpong

第二个练手程序叫做 pingpong，是一个典型的进程通信应用。它的运行逻辑是：父进程使用 fork()系统调用，创建一个子进程，然后使用管道 pipe，在两者之间进行通信。例如：

(1) 父进程创建子进程；

(2) 父进程用管道向子进程发送信息（ping）；

(3) 子进程收到此信息后，回复父进程一个信息（pong）；

(4) 完成上述动作后，两者都退出。

提示：除了 fork 系统调用，上述过程还需要用到创建管道的系统调用 pipe。大家之前如果没有用到的话，可以去网络上查一下它们的基本用法。虽然 xv6 对这些标准的系统调用的实现会和 Linux 有所不同，但是接口的使用方法基本一致。此外，父进程与子进程可能需要一个同步的动作，以免提前退出。

程序编写完成后，参考 1.4.1 的程序编译部分，把 pingpong 程序放到对应的 xv6 源码目录，然后修改 Makefile 文件，再编译运行。

1.4.3 primes

这里的 prime 是质数的意思。该程序的逻辑是把 2-35 之间的质数筛选出来，并在终端进行输出。看起来不是很难，但是本题目为了和 OS 的知识点进行结合，做出了以下解题步骤约定：

(1) 父进程 A 生成[2-35]之间的所有数字，包括 2 和 35；

(2) 向 A 的子进程 B 进行输出（通过管道）；

(3) B 对[2-35]进行筛选，打印出第一个质数 2；

(4) B 把剩下的数字（通过）通过管道传递给 B 的子进程 C；

(5) C 打印出第二个质数，也即 3；

(6) 重复（4）和（5）的过程，直到 2-35 之间的质数全部在终端输出，如图

1.3 所示。

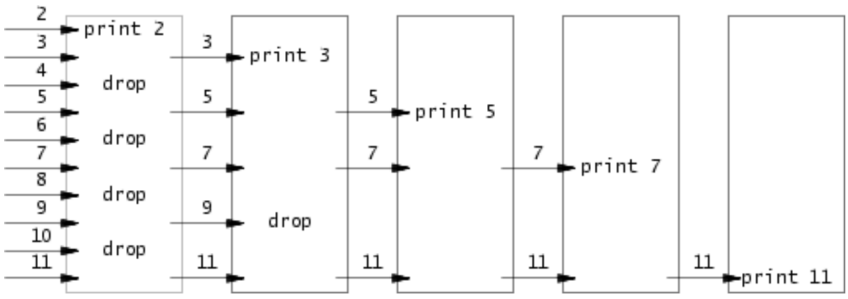

图 1.3 prime 的筛选质数的过程，当中的矩形表示进程，而从左至右的递进关系表示的是父与子关系（左父右子）。

一些提示：

Some hints:

- Be careful to close file descriptors that a process doesn't need, because otherwise your program will run xv6 out of resources before the first process reaches 35.
- Once the first process reaches 35, it should wait until the entire pipeline terminates, including all children, grandchildren, &c. Thus the main primes process should only exit after all the output has been printed, and after all the other primes processes have exited.
- Hint: read returns zero when the write-side of a pipe is closed.
- It's simplest to directly write 32-bit (4-byte) ints to the pipes, rather than using formatted ASCII I/O.
- You should create the processes in the pipeline only as they are needed.
- Add the program to UPROGS in Makefile.

1.4.4 提交实验报告

1．请大家在规定的时间内提交实验报告（PDF 格式），不允许任何形式的迟交；

2．内容包括：xv6 的环境搭建过程、三个程序的详细解题思路、代码和代码注释（代码用黑色文本框包含，注意版面的可读性和整洁性）；

3．整个实验报告最后，还需要一个详细的实验主观心得，口语化也没有关系——因为 272 位同学提交上来的实验报告大都千篇一律，但是有趣的灵魂也会有一些。