

操作系统课程设计报告

高翔宇 201700121149, 荆嘉政 201705130115

2020 年 1 月 9 日

目录

I 要求回顾	4
1 实验内容与任务	5
1.1 任务	5
1.2 工作内容	6
2 实验过程及要求	7
2.1 实验前一学期	7
2.2 实验学期	7
2.2.1 第 1-4 周, 16 课时	7
2.2.2 第 5-6 周, 8 课时	7
2.2.3 第 7 周, 4 课时	7
2.2.4 第 8-11 周, 16 课时	8
2.2.5 第 12-14 周, 12 课时	8
2.2.6 15-16 周, 8 课时	8
3 相关知识及背景	9
4 教学目的	10
5 实验原理及方案	11
5.1 实验的总体思路	11
5.2 实验可能采用的关键技术	12
6 实验报告要求	13
7 考核要求与方法	14

8 参考文献	15
II 实验报告	16
9 Linux0.11 系统源代码分析	17
9.1 引导/启动/初始化	17
9.1.1 引导器——bootsect.s	17
9.1.2 初始化 CPU 和其他硬件——setup.s	17
9.1.3 初始化 c 语言运行环境——head.s	18
9.1.4 初始化内核功能——main.c	18
9.2 运行	19
9.2.1 系统调用	19
9.2.2 内存管理	19
9.2.3 进程调度	19
9.2.4 进程通信	20
9.2.5 文件系统	20
10 系统运行过程的形式化描述方法	22
10.1 描述	22
10.2 细节	23
10.3 界面与操作方法	23
11 可视化方法的描述	28
11.1 使用的工具	28
11.2 工作原理	28
11.3 实现功能	29
11.4 程序架构	29
11.5 各个模块的实现原理	29
11.6 最终效果以及相关截图	30
11.7 编译、配置、运行的具体方法	35
12 内核运行数据输出方法	38
12.1 概况	38
12.2 gdb 脚本提取源数据	38

12.3 源数据到中间数据和 json	39
13 分工说明	40
13.1 工作部分	40
13.2 实验报告部分	40

I

要求回顾

1

实验内容与任务

1.1 任务

该实验以 Linux0.11 为例帮助学生探索操作系统的结构、方法和运行过程，理解计算机软件和硬件协同工作的机制。学生需要完成 4 项任务：

- (1) 分析 Linux0.11 系统源代码，了解操作系统的结构和方法。
- (2) 通过调试、输出运行过程中关键状态数据等方式，观察、探究 Linux 系统的运行过程。
- (3) 建立合适的数据结构，描述 Linux0.11 系统运行过程中的关键状态和操作，记录系统中的这些关键运行数据，形成系统运行日志。
- (4) 用图形表示计算机系统中的各种软、硬件对象，如内存、CPU、驱动程序、键盘、中断事件等等。根据已经产生的系统运行日志，以动画的动态演示系统的运行过程。

1.2 工作内容

将整个系统的运行过程可视化需要付出巨大的工作量，一个学期内难以完成。在全面分析源代码的基础上，学生可以根据自身的能力和兴趣在不同层次、规模、难度上完成本项实验：

- (1) 学生可以探究系统某个模块的某个过程，如文件系统的读操作、键盘的输入、CPU 的调度等。
- (2) 学生可以选择组成大小不等的团队参与实验。
- (3) 在可视化处理上，学生也可以做适当的简化。

2

实验过程及要求

2.1 实验前一学期

在操作系统原理课程中，教师介绍 Linux0.11 源码结构及相关资料，并公布下一学期操作系统课程设计的任务。学生具备了自己分析源代码的基础。

2.2 实验学期

2.2.1 第 1-4 周，16 课时

将 Linux0.11 源代码分成基础模块和选读模块，学生必须分析基础模块，然后从选读模块中选择感兴趣的模块重点分析。

2.2.2 第 5-6 周，8 课时

学生自由组合成团队，提出设计方案，每个团队说明感兴趣的系统运行过程。

2.2.3 第 7 周，4 课时

讨论、评估设计方案。

2.2.4 第 8-11 周, 16 课时

从感兴趣的系统运行过程中提取系统运行的状态数据，并生成系统运行日志。

2.2.5 第 12-14 周, 12 课时

根据日志实现运行过程的可视化。

2.2.6 15-16 周, 8 课时

学生演示运行结果，评定成绩。

3

相关知识及背景

实验以 Linux 操作系统为背景，涉及

- 操作系统原理 (80%)
- 计算机组装 (30%)
- 计算机体系结构 (30%)
- C 语言 (80%)
- 数据结构 (30%) 等课程中的基本知识和方法

通过实验学生的如下方面的能力将得到训练和发展

- (1) 代码分析能力
- (2) 编程能力
- (3) 计算机系统能力
- (4) 沟通协作能力
- (5) 表达能力

4

教学目的

- (1) 将操作系统原理与具体实现相结合，加深对理论知识的理解
- (2) 掌握计算机系统的硬软件整体架构，培养学生的全局观和系统能力
- (3) 理解运行中的系统，锻炼学生解决实际问题的能力

5

实验原理及方案

该实验是一个综合性很强的课程设计，包含了计算机系统中硬件和软件设计的多项基本原理：

- CPU 结构
- CPU 管理
- 内存管理
- 外设控制
- 文件系统

5.1 实验的总体思路

- (1) 在代码级别上理解系统的运行过程；
- (2) 获取系统运行过程的数据；
- (3) 演示系统运行的过程。该实验的结果是开放的，解决具体问题的思路依赖于学生各自的设计目标。

5.2 实验可能采用的关键技术

内核编程技术，用以从内核中输出运行时数据，需要修改内核。其难点在于新增加的代码应该保持内核运行时原有的样子，而且内核编程与普通编程相比，受到更多的限制。

可视化技术，即如何利用图形图像表达系统的运行过程。

6

实验报告要求

- (1) Linux0.11 系统源代码分析报告，说明学生本人分析代码的体会及重点分析的代码描述
- (2) 系统运行过程的形式化描述方法
- (3) 内核运行数据输出方法
- (4) 可视化方法的描述
- (5) 系统运行过程实例（图文描述）

7

考核要求与方法

节点	课序	标准	考核方法
设计方案	28	创新性；完整性；可行性。30%	课堂报告，教师打分
系统运行 过程描述	44	系统状态的形式化描述； 状态数据的获取技术。30%	课堂报告，教师打分
演示	60	表现力；流畅性；界面。30%	课堂报告，教师打分
实验报告	64	规范性；文字表达。10%	教师打分

8

参考文献

- Linux 内核完全解析 0.11
 - 赵炯
- Linux 内核设计的艺术
 - 新设计团队

II

实验报告

9

Linux0.11 系统源代码分析

9.1 引导/启动/初始化

9.1.1 引导器——bootsect.s

bootsect.s 将 setup.s 代码和 system 模块加载到内存中，并且分别把自己和 setup.s 代码移动到物理内存 0x90000 和 0x90200 处后，就把执行权交给了 setup 程序。其中 system 模块的首部包含有 head.s 代码。

9.1.2 初始化 CPU 和其他硬件——setup.s

setup 程序的主要作用是利用 BIOS 的中断程序获取机器的一些基本参数，并保存在 0x90000 开始的内存块中，供后面程序使用。同时把 system 模块往下移动到物理地址 0x00000 开始处，这样，system 中的 head.s 代码就处在 0x00000 开始处了。然后加载描述符表基地址到描述符表寄存器中，为进行 32 位保护模式下的运行作好准备。接下来对中断控制硬件进行重新设置，最后通过设置机器控制寄存器 CR0 并跳转到 system 模块的 head.s 代码开始处，使 CPU 进入 32 位保护模式下运行。

9.1.3 初始化 c 语言运行环境——head.s

Head.s 代码的主要作用是初步初始化中断描述符表中的 256 项门描述符，检查 A20 地址线是否已经打开，测试系统是否含有数学协处理器。然后初始化内存页目录表，为内存的分页管理作好准备工作。最后跳转到 system 模块中的初始化程序 init/main.c 中继续执行。

9.1.4 初始化内核功能——main.c

main.c 程序首先利用前面 setup.s 程序获得的系统参数设置系统参数设置系统的根文件设备号以及一些内存全局变量去分配内存，之后内核进行硬件初始化工作。完成内核初始化后，CPU 从内核态变为用户态。之后系统创建一个新进程用于运行 init() 子进程。init 子进程主要完成了安装根文件系统，显示系统信息，运行系统初始资源配置和执行用户登录 shell 程序的四个功能。

代码前 40 行引入头文件，42 行到 53 行完成外部函数的定义，主要是设备和硬件的初始化函数，58, 59, 60 三行宏定义了 1MB 以后的扩展内存大小，硬盘参数表的 32 字节内容，根文件系统所在设备号，将这三个线性地址转换为给定数据类型的指针，并获取指针所指内容。69 行到 72 行实现了读取 CMOS 实时时钟信息，74 行实现了将 BCD 码转为二进制。76 行到 96 行获取 CMOS 中开机时间并将其转换为二进制。接下来定义了一些局部变量并进入主函数。主函数首先保存根设备号，高速缓存末端地址，机器内存数，主内存开始地址，实现过程为 110 行到 122 行，主内存开始地址等于缓冲区末端。126 行到 136 行实现了内核所有方面的初始化工作。137 行将程序从内核态切换到用户态进行工作，接下来并在新建的子进程中执行 init()。151 行到 161 行实现了我们常用的 printf() 函数，使用 vsprintf() 将格式化的字符串放入 printbuf 缓冲区，然后用 write() 将缓冲区的内容输出到标准设备。接下来 168 行到最后实现了 init 函数。init 函数以读写访问的方式打开设备 “/dev/tty0”，即中断控制台，接下来到 178 行打印缓冲区块数和总字节数，以及主内存区空闲内存字节数。程序 180 行到 185 行实现了将自身进程替换到 shell 程序，并将 stdin 重定向到/etc/rc 文件，以运行 rc 中设置的命令。至此 main.c 文件分析完毕

9.2 运行

9.2.1 系统调用

当用户态进程发起一个系统调用，CPU 将切换到内核态并开始执行一个内核函数。内核函数负责响应应用程序的要求，例如操作文件、进行网络通讯或者申请内存资源等。具体进行调用的流程如下：应用程序代码调用系统调用 *xyz*，该函数是一个包装系统调用的库函数；库函数 *xyz* 负责准备向内核传递的参数，并触发软中断以切换到内核；CPU 被软中断打断后，执行中断处理函数，即系统调用处理函数 *system_call*；系统调用处理函数调用系统调用服务例程 *sys_xyz*，真正开始处理该系统调用；

9.2.2 内存管理

在 8086CPU 中，程序寻址使用的是由段选择符和偏移地址构成的地址，这个地址并不能直接寻址物理内存，因此被称为虚拟地址。

虚拟（逻辑）地址首先通过分段管理机制首先转换成 CPU32 位线性地址（称作中间地址）

实际函数中操作的地址为当前进程数据段的段内偏移，然后会根据该段的段基址和段内偏移确定一个线性地址然后利用分页管理机制将此线性地址映射到物理地址。

分页机制会将线性地址分为页目录项、页表项、和页内偏移三部分。利用这三部分确定该线性地址指向的实际物理地址。在 linux0.11 中只使用了一个页目录表，当前页目录表用寄存器 CR3 来确定。

9.2.3 进程调度

进程调度中的一个重要函数就是 *fork.c* 子程序，该程序是 *sys_fork()* 系统调用的辅助处理函数集。

首先前 20 行是头文件的引用，接下来 24 到 28 行实现了空间区域先前验证函数，该函数首先将起始地址调整为其所在页的左边界开始地址，然后通过循环验证每个页面是否可写来确定该内存空间是否可写。接下来 *copy_mem* 函数，首先对代码段和数据段的限制和基地址进行初始化，然后比对代码段和数据段基地址是否相等以及地址限长是否相同，否则

报错。检查完毕后设置创建的新进程在线性地址空间的基地址为 64MB 乘其任务号，并将该值设置为新进程局部描述符中的基地址，接着复制页表，如果返回值不为 0，则释放刚才申请的页表项。接下来为复制进程的函数。首先为新任务分配页，如果分配页出错，则返回-EAGAIN 报错。然后将新任务的结构指针放入对应任务数组的 nr 项中，nr 为任务序号，之后将 current 的内存空间拷贝给 p 进行初始化，接下来对 p 的 state, pid 等属性——修改，之后 99 行到 112 行修改任务状态段 tss 数据。接下来调用之前实现过的 copy_mem 函数复制进程页表。如果父进程中文件是打开的，则对应文件的打开次数就加一，pwd, root, executable 这些也同理。之后在 GDT 表中设置 tss 和 ldt，最后将任务设置为就绪态，以防万一，最后返回进程的 pid。最后一个函数是返回任务数组中的任务号的函数，该函数实现过程为，首先获取一个新的进程号，如果 last_pid 加一后超出进程号的正数表示范围，则重新从 1 开始使用 pid 号。然后在任务数组中搜索刚设置的 pid 号是否可用，最后如果所有任务都被占用即 NR_TASKS 个任务都被占用，则返回-EAGAIN 报错。

9.2.4 进程通信

在 linux0.11 中进程间通信主要是通过管道实现的，所谓管道，是指用于连接一个读进程和一个写进程，以实现它们之间通信的共享文件，又称 pipe 文件。向管道（共享文件）提供输入的发送进程（即写进程），以字符流形式将大量的数据送入管道，而接收管道输出的接收进程（即读进程），可从管道中接收数据。由于发送进程和接收进程是利用管道进行通信的，故又称管道通信。这种方式能够传送大量数据，且很有效。在 Linux 中，管道的实现并没有使用专门的数据结构，而是借助了文件系统中 file 结构和 VFS 的索引节点 inode。通过将两个 file 结构指向同一个临时的 VFS 索引节点 inode，这个索引节点又指向同一个物理页面来实现的。管道实现的源码在 fs/pipe.c 中，pipe.c 中只有三个函数，write_pipe()、read_pipe() 和 sys_pipe()。

9.2.5 文件系统

0.11 版的内核中采用了 minix1.0 版的文件系统。minix 文件系统和标准 unix 文件系统基本相同。它由 6 个部分组成，分别是：引导块，

超级块，*i* 节点位图，逻辑块位图，*i* 节点，和数据区。如果存放文件系统的设备不是引导设备，那么引导块可以为空。PC 机的块设备通常以 512 字节为一个扇区，而文件系统则以盘块为单位使用之。*minix* 中 1 个盘块等于 2 个扇区大小。从引导块为第 0 个盘块开始计算。逻辑块可以为 2^n 个盘块，*minix* 中逻辑块大小等于盘块。所以盘块 = 逻辑块 = 缓冲块 = 1024 字节。超级块存放文件系统的整体信息。*i* 节点位图描述了 *i* 节点的使用情况。文件通常将控制信息和数据分开存放，*i* 节点就是存放文件的控制信息的，通常称之为 *inode*。逻辑块位图则描述了逻辑块的使用情况。*linux* 中的文件范围很广泛，不仅仅指普通文件。用 `ls -l` 命令可以发现显示的信息的最左边字符可以为“-”，“d”，“s”，“p”，“b”，“c”，分别表示正规文件，目录文件，符号连接，命名管道，块设备，字符设备文件。紧跟在其后的 9 位字符可以为 r,w,x,s,S 等，描述了文件的访问权限。后面的信息有文件的用户名，组名，文件大小，修改日期等，这些信息当然是放在 *inode* 中的。文件名除外。那么文件系统是如何被加载的呢？在系统启动过程中，具体是在任务 1 的 `init()` 函数中，通过 `setup` 系统调用加载的，该函数调用 `mount_root()` 函数读取根文件系统的超级块和根 *inode* 节点。

10

系统运行过程的形式化描述方法

10.1 描述

我们主要对 exec.c 程序中的 do_execve() 函数进行可视化实现，该函数是系统调用 (int 0x80) __NR_execve() 调用的 c 处理函数，是 exec() 函数的内核实现函数。该函数首先执行对命令行参数和环境参数空间页面进行初始化，首先设置空间起始指针并且将页面指针数组置为空，之后根据执行文件名取执行对象的 i 节点，之后调用 count 函数计算参数个数和环境变量个数，之后检查文件类型和执行权限。

初始化完成后，根据执行文件开始部分的执行头，将其中信息进行处理。如程序开头是“#!”则分析 shell 程序名及其参数，并按照 shell 程序进行执行。根据文件的 magic number 以及段长度信息判断程序是否可以执行。

之后进行程序执行前的初始化操作。将当前指针指向新执行文件的 inode，复位信号处理句柄，并根据头结构信息设置局部描述符基址和段长，设置参数和环境参数页面指针，修改进程各执行字段内容，最后替换堆栈上原调用 execve() 程序的返回地址为新执行程序的运行地址，运行新加载的程序。

我们组工作的两点在于我们没有将操作系统中一个设计很多源程序的

大动作进行展示，而是重点展示一个程序的运行过程，数据详细，动画精炼。和计划书中相比，我们对自己的实验完成度很满意，计划书中提到的内容我们在实验过程中都进行了实现，还对计划书中的细节进行了完善优化。

10.2 细节

(此处参考计划书第一部分的“细节”小节的要求填写，与计划书不同的是，此处应该填写的不是计划而是最终实现效果的相关内容，注意表格新增最后一列完成度)

实现表格如下

10.3 界面与操作方法

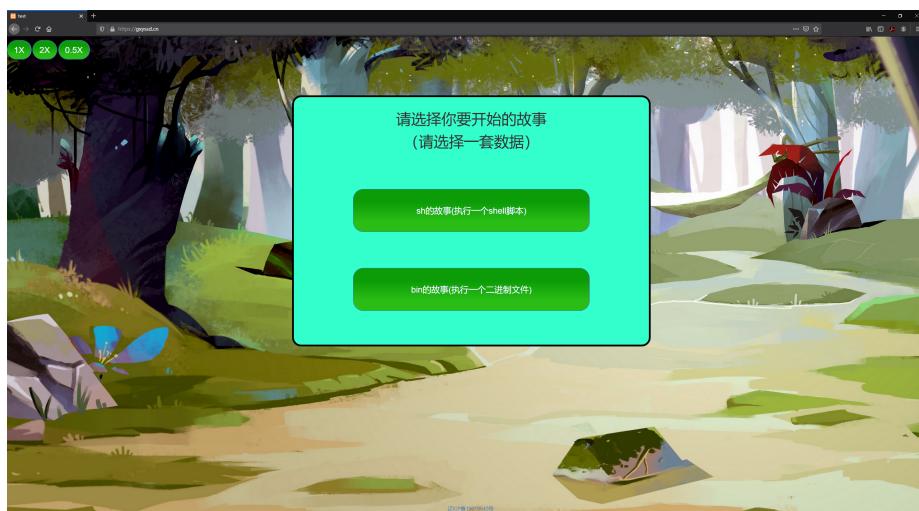


图 10.1：可视化操作界面

用户可以通过左上角的三个按钮调整动画播放的速度，此外，在刚打开页面的主页面上面用户可以通过这两个按钮选择不同的数据进行执行，不同的数据将会产生不同的动画。在我们这里，我们分别进行了 `ls` 命令的执行和 `shell` 脚本的执行进行了数据的提取，根据这些数据生成了两套不同的动画，用户可以通过这两个按钮自由选择。

编 号	动作	事件	完 成 度
1	(命令行界面) 用户输入命令, 例如“ls”	执行过程开始	√
2	(显示 sys_call 代码段) 在 202 行调用 do_execve	字符解析等过程后, 进入系统调用, 执行到 kernel/sys_call:202 处时调用 do_execve	√
3	(进入程序执行界面) 读取 eip[1]	加载原用户程序代码段寄存器 CS 值	√
4	(判断动画) 比较 0xffff & eip[1] 与 0x000f	判断上一步中的 CS 值是否是当前任务的代码段选择符	√
5	(调用程序动画) 读取 namei(filename)	读取当前程序结点指针	√
6	(判断动画) 比较 inode 和 namei(filename)	判断当前执行程序 i 节点是否与内存中 i 节点指针相同	√
7	读取 inode 的 i_mode 属性	读取 i_mode 属性	√
8	读取 e_uid 并移位	读取用户 id 准备进行权限检查	√
9	读取 e_gid 并移位	读取组 id 准备进行权限检查	√
10	与 i_mode 做运算	检查是否有权执行目标文件	√
11	(没有权限)(切换到错误界面) 跳入 error2	权限不足, 程序结束	√
12	(有权限) 读取 ex=exec-header	读取程序执行头	√
13	(判断动画) 判断 ex 是否以 '#!' 开头	判断目标程序是否是脚本程序	√
14	(是脚本, 切换界面) 读取脚本解释段	将脚本解释段加载到变量 buf 中	√
15	释放缓冲块, 放回脚本 i 节点	执行 brelse(bh)	√
16	(动画) 把 i 节点放入环境参数块	执行 iput(inode)	√
17	(动画) 对 buf 字符串进行清洗, 去掉开始的空格和制表符	通过一系列字符串处理函数对原字符串进行清洗	新增
18	(判断动画) 判断 buf 第一行是否为空	取出 buf 第一行的内容判断是否为空	√
19	(为空)(切换到错误界面) 置错误码, 跳入 error1	修改 retval 值, 执行 goto 命令到 error 处理部分	√
20	(不为空继续执行) 读取解释程序的文件名和参数	读取 i_name 和 i_arg 内的值到变量	√

21	打标记，防止函数反复重启	执行 sh_bang++, 将标记变量 +1	√
22	进入 copy_strings 申请空间 载入环境变量	执行 copy_strings(envc, envp, page, p, 0)	
23	(切换界面进入 copy_strings 函数)	跳入 copy_strings 函数 (首先把参数 复制到偏移地址中)	√
24	读取指向内核数据段的 ds 值	执行 new_fs	部分完成
25	读取当前段寄存器 fs 值	= get_ds(), 把内核数据段 ds 存入 new_ds	部分完成
26	(判断动画) 判断 from_kmem 是否等于 2	执行 old_fs	部分完成
27	(在内核空间)(动画)fs 指向内 核空间	= get_fs(), 同上一步, 把 fs 保存, 方便以后在内存态和用户态之间切换	部分完成
28	(进入循环)(循环到 36 步结 束)	判断参数串及其指针是否在内核空间	√
29	(判断动画) 判断 from_kmem 是否等于 1	改变 fs 指向内容, 若参数串和指针在内 核空间则 fs 进入内核空间	√
30	(在内核空间)(动画)fs 指向内 核空间		√
31	读取字符串指针	判断指针是否在内核空间	√
32	(判断动画) 判断 from_kmem 是否等于 1	改变 fs 指向内容, 若指针在内核空间则 fs 指向内核空间	√
33	(在内核空间)(动画)fs 指向内 核空间	使用 get_fs_long 把指针读到 fs 处	√
34	读取参数字符串	判断串及其指针是否在内核空间	√
35	(判断动画) 剩余空间是否小于 0	改变 fs 指向内容, 如果指向内核空间则 指向用户空间	√
36	(小于零)fs 指向用户空间, 直 接退出, 不进入 error 处理程 序	使用 get_fs_byte 逐个读取字符	√
37	(进入循环)	计算指针位置 p-存储区间大小 len, 若 小于零则异常	√

38	找到 p 指针在页面内的偏移值	先修改 fs 位置后直接 return0(已经复制到偏移地址中，下一步复制到参数和环境空间)	√
39	(判断动画) 判断 from_kmem 是否等于 1		√
40	(在内核空间)(动画)fs 指向内核空间	把偏移值存入 offset	√
41	(判断动画)p 指针所在页面是否存在	判断串及其指针是否在内核空间	√
42	(若不存在) 调页动画	把 fs 的位置复原，回到内核空间	√
43	(若申请不到页面) 直接退出程序	调用 p 所在的串空间页面指针数组项，查看当前页面是否存在	√
44	(判断动画) 判断 from_kmem 是否等于 2	执行 get_free_page() 函数来申请页面	√
45	(在内核空间)(动画)fs 指向内核空间	直接执行 return	√
46	读取 offset 和 pag 的代表的地址	0	√
47	从 fs 复制 1 字节到 page 的 offset 处	判断指针是否在内核空间	√
48	(判断动画) 判断 from_kmem 是否等于 2	改变 fs 指向内容，若字符串在内核空间则 fs 指向内核空间	√
49	(在内核空间)(动画)fs 指向内核空间	返回 pag+offset	√
50	(copy_strings 函数结束) 返回参数和环境空间中已复制参数的头部偏移值	执行 get_fs_byte 移动一个字节	√
51	(返回 exec.c 执行界面) 之前为复制环境字符串，后重复 23-50 步复制命令行参数.	判断字符串和字符串数组是否在内核空间	√
52	后重复 23-50 步复制脚本文件名	把 fs 的位置复原	√
53	后重复 23-50 步复制解释文件程序名	执行 return	√
54	后重复 23-50 步复制解释文件参数		√
55	(判断动画) 判断 p 是否为 0	查看上述复制字符串的页面是否被填满	√
56	(p=0) 页面已满，置错误码，跳 error1	给 retval 复制，执行 goto 语句	√

57	保存原 fs 段寄存器设置后 fs 指向内核段	先赋值后指向内核段	√
58	取得解释程序的 i 节点	使用 namei 函数	√
59	(出错) 置错误码, 返回 error1	查看上述复制字符串的页面是否被填满	√
60	fs 指回用户段	给 retval 复制, 执行 goto 语句	√
61	(跳到程序开始界面) 返回前部分循环执行	先赋值后指向内核段	√
62	(接 13 步, 不为脚本) (动画) 释放 bh 缓冲块	使用 namei 函数	√
63	(长判断动画) 判断目标文件是否可执行		√
64	(若不能执行)(跳错误界面) 置错误码, 跳 error2	复原 fs 的值	√
65	(判断动画) 判断环境变量页面是否已经复制	使用 goto 语句返回 restart 节点	√
66	(若未复制) 复制命令行参数和环境字符串	调用 brelse(bh) 释放 bh 块	√
67	(若无空间) (跳错误界面) 置错误码, 跳 error2	对 ex 中的执行头信息进行一系列字符串处理, 判断能否执行	√
68	(动画) 返回原执行程序 i 节点并使 executable 指向新执行文件的 i 节点	给 retval 复制, 执行 goto 语句	√
69	逐一关闭文件, 更改文件句柄	直接读取 sh_bang 中的内容	√
70	根据当前进程的地址和限长释放原程序的代码段和数据段对应的内存页表	同上述过程, 调用 copy_strings 分别复制命令行参数和环境字符串	√
71	根据之前加载的字符串长度计算出栈指针位置	给 retval 复制, 执行 goto 语句	√
72	(动画) 在栈空间中创建环境和参数变量指针表	(这里开始准备释放当前进程占用的系统资源, 关闭文件, 释放内存, 并申请新内存空间)	√
73	修改 current 各字段的值为新执行文件的信息	将 current 中存放的当前进程打开的文件关闭	√
74	原系统中断在堆栈上的代码指针替换为新执行程序的入口点	调用 free_page_tables 释放页面	√
75	栈指针替换为新程序的栈指针 p	页面最高位减去字符串总长即为栈指针 p 的位置	√
76	(错误界面 error2) 放回 i 节点	调用 create_tables	√

11

可视化方法的描述

11.1 使用的工具

可视化部分的开发使用的语言为 `html, javascript, css`, 使用到的框架包括 `bootstrap, jquery`, 以及一个基于 `js` 的动画库—`anime.js`, 前两个库使用的是 `staticfileCDN` 所提供的在线资源, 而 `anime.js` 使用的是本地的扩展包, 该扩展包已经和源码进行整理合并上传。最终实现的效果为一个网页动画的形式。

11.2 工作原理

该动画主要通过 `anime.js` 框架实现, 该框架动画代码格式如下:

代码 11.1 动画代码

```
anime({
  targets: '.target1,.target2',//动画的执行目标为 target1, target2
  translateX:200,           //表示在X方向上移动到200像素的位置。
  duration:1000,            //表示整个动画执行过程为 1000ms。
  delay:2000,               //表示该动画延迟2000ms执行。
  complete:function(){
    //里面定义了该动画执行完毕后调用的过程，可以是方法或动画。
  }
});
```

11.3 实现功能

该部分可视化程序实现的主要功能为一个动画生成器，根据我们事先约定好的 json 格式，我对 json 内的数据进行解析，解析之后根据这些数据，决定播放哪个类型的动画。json 数据中的一条便定义了一个动作，因此该可视化程序，不仅仅可以用于生成我们所做的 do_execve() 源码的展示，只要有其他代码部分的数据提取，该动画生成器一样可以根据这些 json 文件进行动画的生成，这是这个动画生成器的最强大之处。

11.4 程序架构

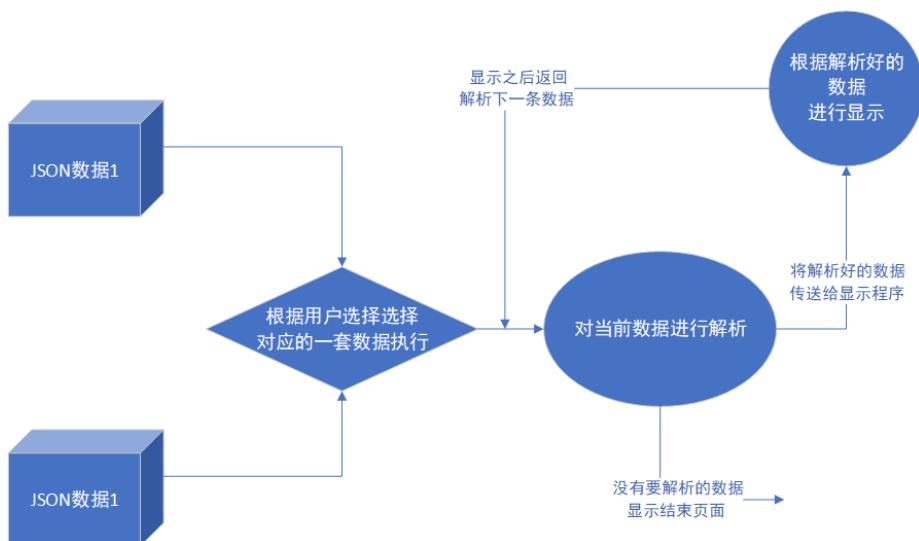


图 11.1： 可视化程序流程图

11.5 各个模块的实现原理

该部分我将介绍一下动画的实现原理以及代码的逻辑，在解析出一条 json 数据之后，我采用 switch 分支结构，通过判断 json 中的动画类型（预先已经协商定义好），跳转到对应的 case 情况之下。由于每一步动

画只有 `complete` 属性之后的函数才会在该动画执行结束后执行，因此我采用递归调用的方式对这个函数进行循环，并没有采用循环语句。其中，我首先在 `html` 文件中把需要的模块都定义好，然后将它们的 `display` 属性都设置为 `none`，即不显示在主界面上。之后，在不同的动画中，根据每个模块标签预先设置好的 `id` 对它们进行访问修改它们的 `css` 属性，将他们显示在界面上。其中，它们的位置和大小也早就在 `css` 文件中设置完毕。之后如果有需要有进入的特效动画，我们则先将这个模块隐藏起来，之后将这个模块移出这个屏幕范围之外，再将其显示出来，再将这个模块移回这个屏幕范围之内，便制作出了进场动画的效果。离场动画，只需要讲对应的模块移出屏幕，之后设置 `display` 属性为 `none` 即可。以上便是这些模块的实现原理。

11.6 最终效果以及相关截图

该程序的运行效果如下图所示：



图 11.2：运行界面图 1

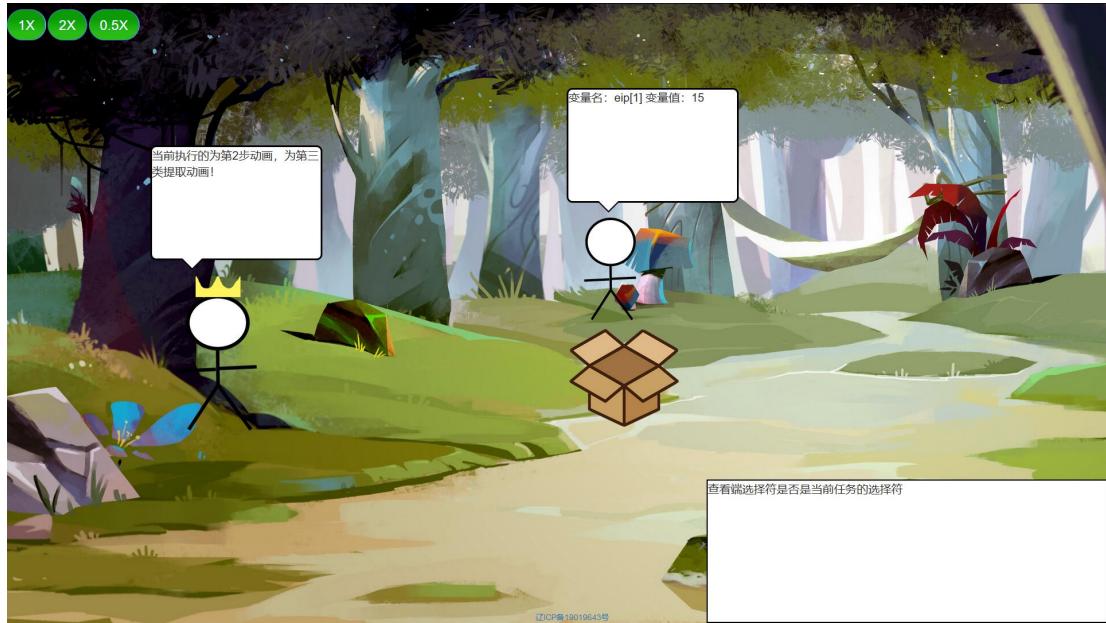


图 11.3：运行界面图 2

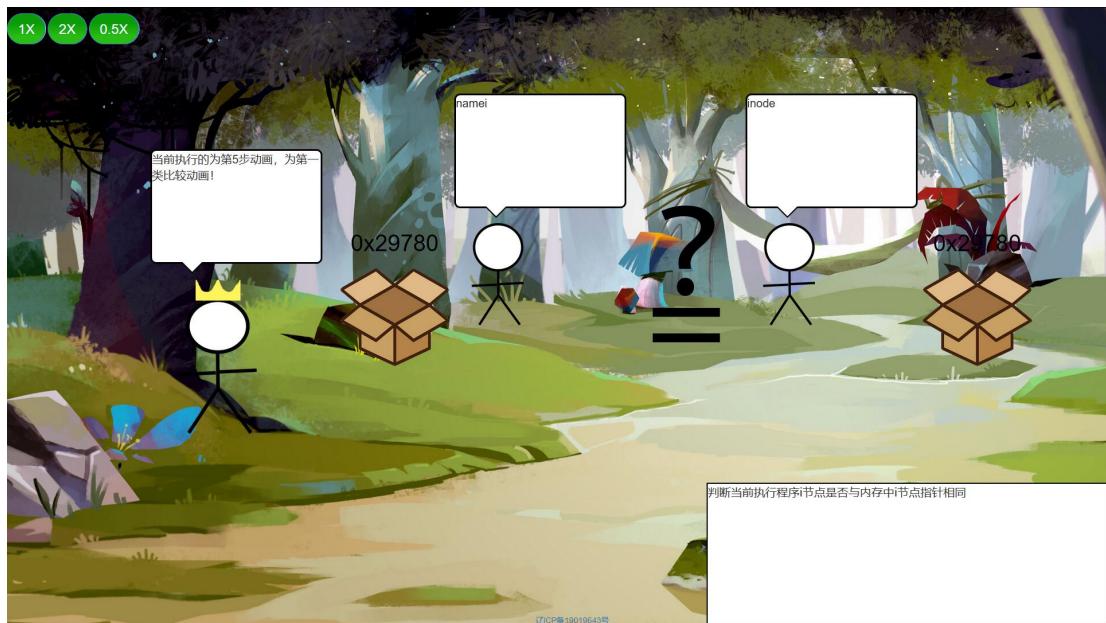


图 11.4：运行界面图 3



图 11.5：运行界面图 4



图 11.6：运行界面图 5

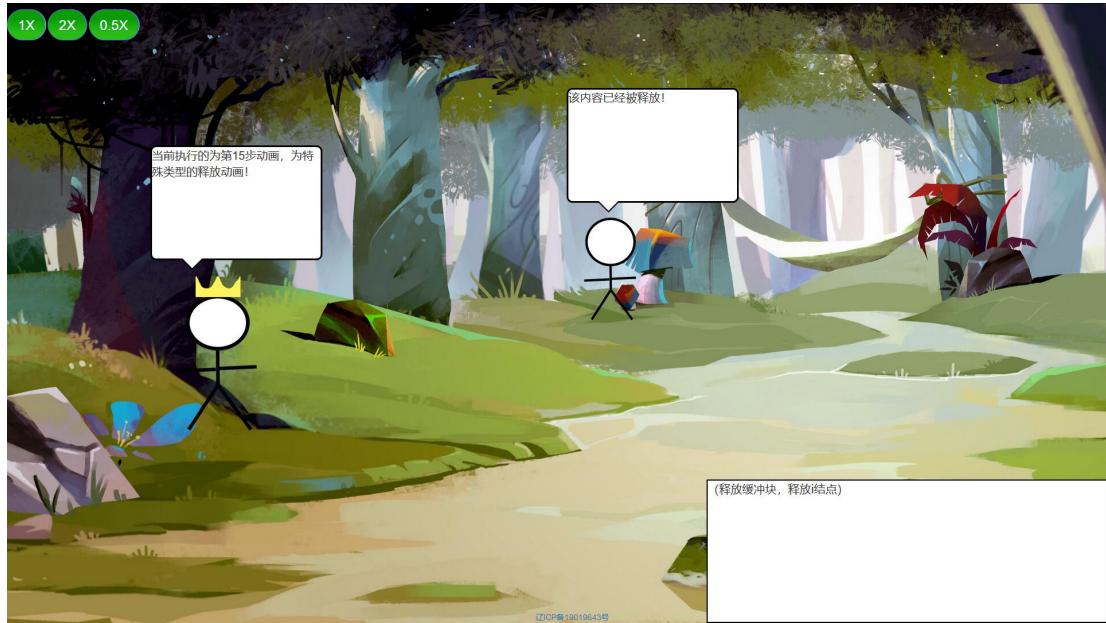


图 11.7：运行界面图 6

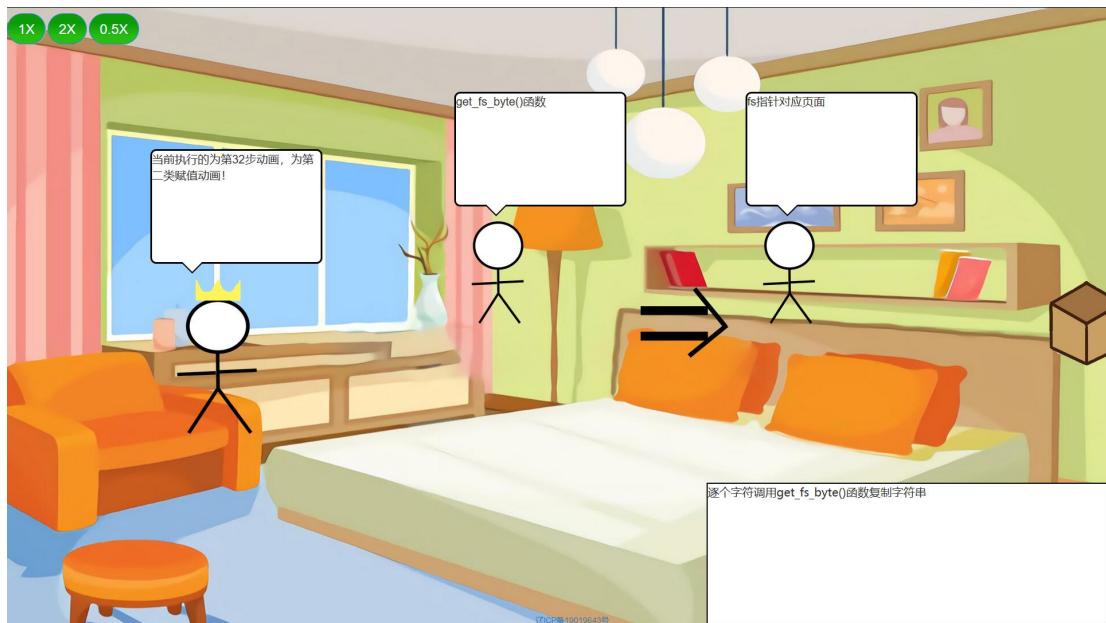


图 11.8：运行界面图 7

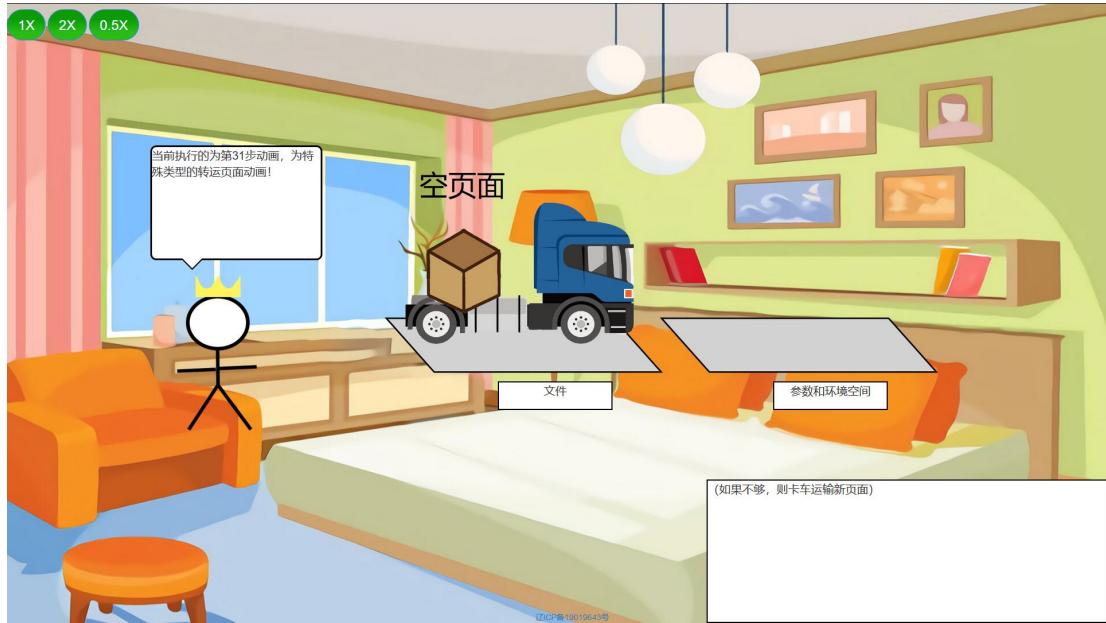


图 11.9：运行界面图 8

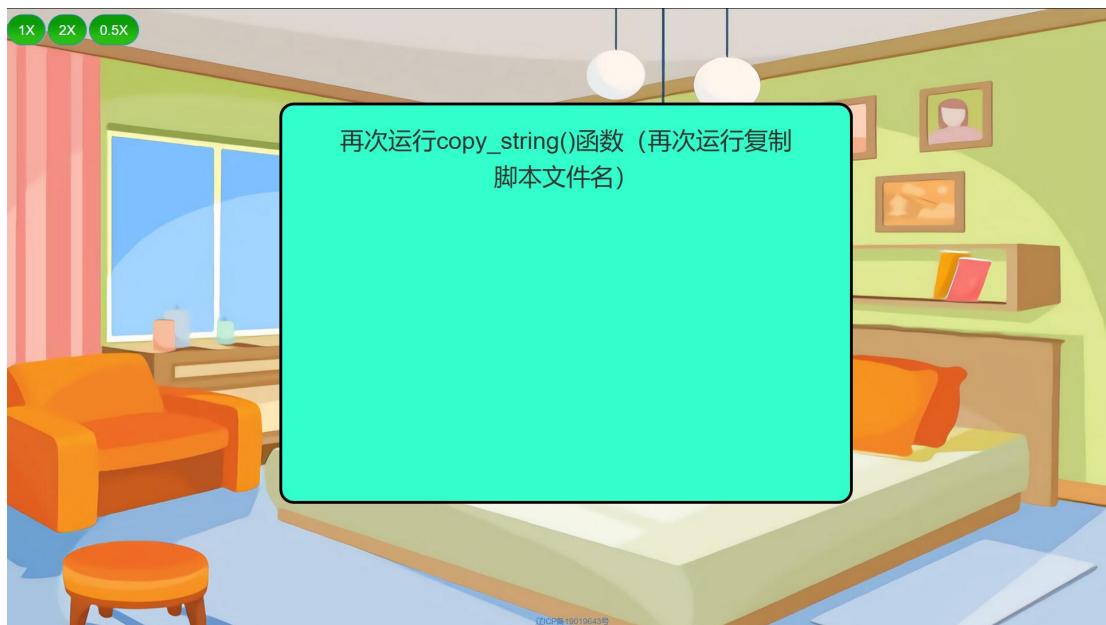


图 11.10：运行界面图 9

11.7 编译、配置、运行的具体方法

首先下载我们的代码，下载链接为https://github.com/gxy666/OS_PRO，接下来下载安装xampp 安装包，下载链接为<https://www.apachefriends.org/index.html>，根据引导安装完毕后，运行该程序。可以看到如下图(xampp 程序运行图) 界面：

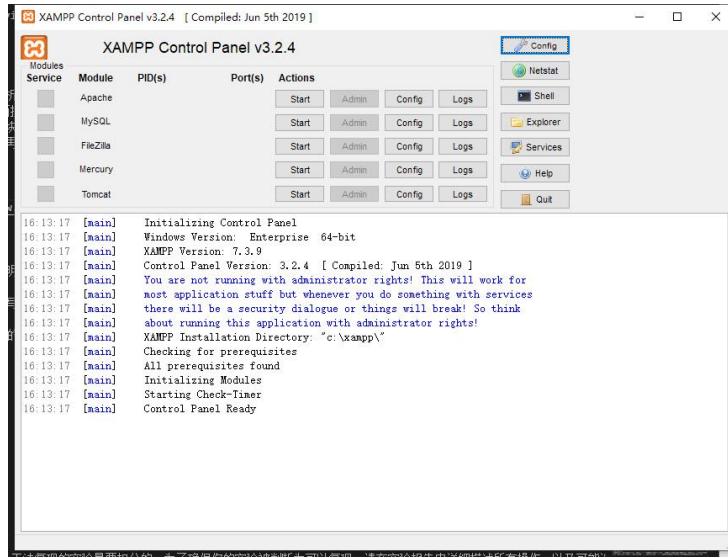


图 11.11: xampp 程序运行图

点击 apache 中的 start 按钮运行服务器，可以看到下面日志输出 running 则为运行成功，(如果运行不成功，请检查电脑的 80 端口的占用情况，可能会和 VMware 等软件冲突) 如下图 (xampp 程序服务器运行成功图) 所示：

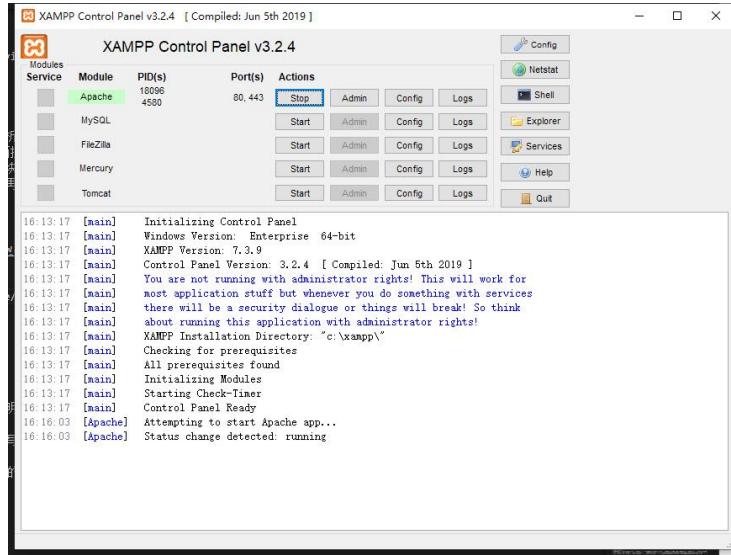


图 11.12: xampp 程序服务器运行成功图

之后点击 config 按钮，选择其中第一个 Apache(httd.conf) 选项(如下图 xampp 程序服务器配置文件打开图所示)，则会打开 apache 服务器的配置文件。

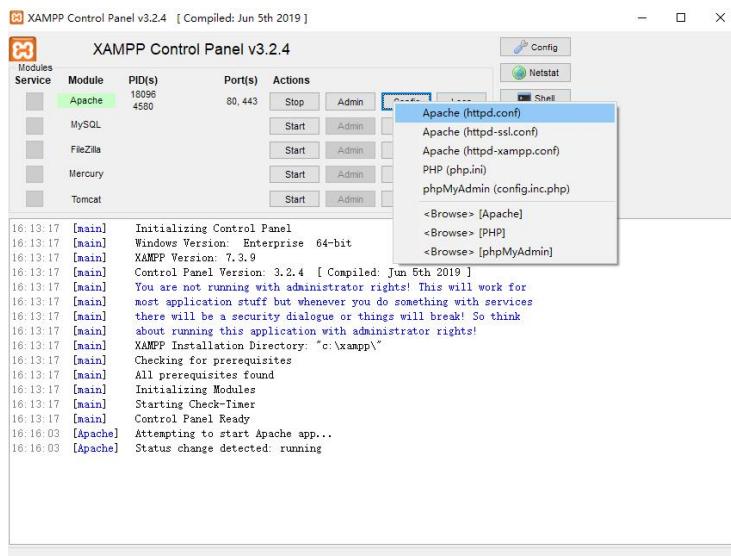


图 11.13: xampp 程序服务器配置文件打开图

找到服务器的如下图（xampp 程序服务器配置文件图）的部分：

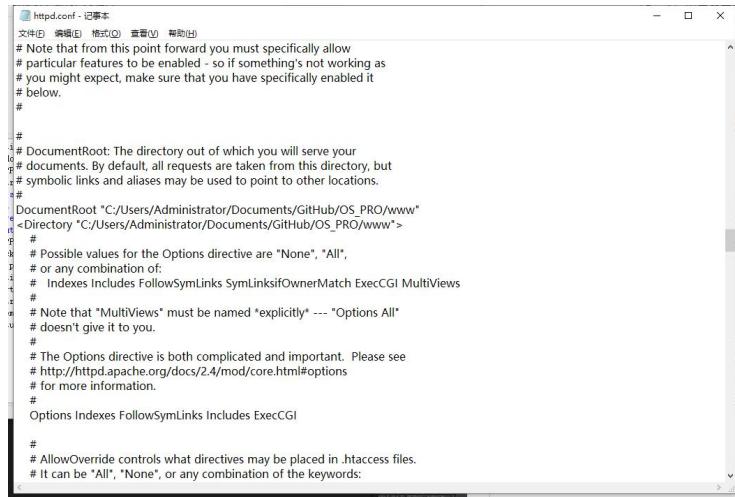


图 11.14：xampp 程序服务器配置文件图

修改里面的 `documentroot` 后面的目录为你下载源码的目录（最好不要有中文），把下面的 `<Directory` 后面的目录也改为对应的目录。该目录默认为你安装 xampp 的目录/xampp/htdocs。你也可以把下载好的源码内容放入该目录下。之后点击 xampp 上 apache 后面的 stop，再次运行 apache。打开一个浏览器，输入 `localhost` 就可以看到我们的主页了。当然，你也可以直接访问<https://gxyssd.cn>来访问我们的主页，不过由于背景图像素过大的原因，该网页加载速度可能有些慢。

（以上为可视化部分的使用说明，数据提取部分的使用说明见 12 章。）

12

内核运行数据输出方法

12.1 概况

直接使用 gdb 脚本进行编辑, 将要提取的数据全部保存在 `gdb_output.txt` 中, 之后从 `gdb_output.txt` 中解析数据。我通过阅读代码和分析断点处输出的变量信息构建了中间代码生成的配置文件 `mid-sh.txt` 和 `mid-nonsh.txt`, 通过运行我编写的 `python` 脚本, 可以实现将中间数据转换为 `json` 格式。

`json` 对象的 `key` 包括数据的类型 (我们计划分为整数, 地址 (包括指针), `bool` 值, 对象 (以结构体为主)), 数据的值, 以及数据的用途备注 (如果是对象的话首先将对象的类型进行标注)。之后我们将把这些文件上传到 https://github.com/gxy666/OS_PRO 中的“linux 数据提取”文件夹中。

12.2 gdb 脚本提取源数据

在 `vmware` 软件中安装并打开 `linux0.11_lab`, 进入 `edit_gdb_script_after_call_hello` 函数, 将里面预设的内容全部删除, 然后打开 `edit_gdb_script_before_boot` 函数, 将附件中的 `gdb` 数据提取.txt 文件中的全部粘贴到编辑器中, 保存文件, 然后运行 `run_gdb_script_hd_console_redirect` 函数, 在 `linux0.11` 中

运行 `ls` 命令，在桌面生成 `gdb_output.txt` 文件

12.3 源数据到中间数据和 json

在上一步我们获得了 `gdb_output.txt`, 接下来我们通过 `py` 脚本，将源数据转换为 `json` 格式，首先打开附件中的 `OS.py` 脚本，将 `mid-sh.txt` 文件和 `gdb_output.txt` 文件放到 `OS.py` 同一目录下后运行脚本，运行脚本只需默认 `package`。

脚本运行完毕后即可在文件夹下找到 `OS-sh.json` 文件，改文件即可作为输入直接送入可视化模块驱动动画的生成。

注意，这里可能会报错找不到文件，只要将脚本文件中的 6-9 行进行修改，把其中的文件路径改为存放以上各个文件的文件夹即可。

13

分工说明

13.1 工作部分

本次实验，荆嘉政主要负责数据处理部分，高翔宇主要负责可视化部分。

荆嘉政工作：

1. 筛选需展示的过程
2. 编写 gdb 脚本提取数据
3. 读取数据并转为 json 格式

高翔宇工作：

1. 将 json 格式的文件读入
2. 模块化的编写上述各中动画场景并组合。
3. 搭建服务器并调试

13.2 实验报告部分

本实验报告由荆嘉政，高翔宇共同完成，荆嘉政完成了数据处理相关章节，即 10,12 章，高翔宇完成了可视化相关章节，即 9,11 章。