操作系统课程设计计划书

第一部分 情景 共 25 分

1. 描述：10 分

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。

1. 细节：15 分

（至少 50 个动作）

本部分左侧“动作”一栏描述在可视化过程中进行的动画，右侧“事件”一栏描述动作执行过程中操作系统内核的操作。

编号	动作	事件
1	（命令行界面）用户输入命令，例“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 kernal/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
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 = get_ds(),把内核数据段 ds 存入 new_ds
25	读取当前段寄存器 fs 值	执行 old_fs = 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
26	(判断动画)判断 from_kmem 是否等于 2	判断参数串及其指针是否在内核空间
27	(在内核空间)(动画)fs 指向内核空间	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
28	(进入循环)(循环到 36 步结束)	
29	(判断动画)判断 from_kmem 是否等于 1	判断指针是否在内核空间
30	(在内核空间)(动画)fs 指向内核空间	改变 fs 指向内容,若指针在内核空间则 fs 指向内核空间
31	读取字符串指针	使用 get_fs_long 把指针读到 fs 处
32	(判断动画)判断 from_kmem 是否等于 1	判断串及其指针是否在内核空间
33	(在内核空间)(动画)fs 指向内核空间	改变 fs 指向内容,如果指向内核空间则指回用户空间
34	读取参数字符串	使用 get_fs_byte 逐个读取字符
35	(判断动画)剩余空间是否小于 0	计算指针位置 p-存储区间大小 len,若小于零则异常
36	(小于零)fs 指回用户空间,直接退出,不进入 error 处理程序	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)

37	(进入循环)	
38	找到 p 指针在页面内的偏移值	把偏移值存入 offset
39	(判断动画)判断 from_kmem 是否等于 1	判断串及其指针是否在内核空间
40	(在内核空间)(动画)fs 指向内核空间	把 fs 的位置复原,回到内核空间
41	(判断动画)p 指针所在页面是否存在	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
42	(若不存在)调页动画	执行 get_free_page()函数来申请页面
43	(若申请不到页面)直接退出程序	直接执行 return 0
44	(判断动画)判断 from_kmem 是否等于 2	判断指针是否在内核空间
45	(在内核空间)(动画)fs 指向内核空间	改变 fs 指向内容,若字符串在内核空间则 fs 指向内核空间
46	读取 offset 和 pag 的代表的地址	返回 pag+offset
47	从 fs 复制 1 字节到 page 的 offset 处	执行 get_fs_byte 移动一个字节
48	(判断动画)判断 from_kmem 是否等于 2	判断字符串和字符串数组是否在内核空间
49	(在内核空间)(动画)fs 指向内核空间	把 fs 的位置复原
50	(copy_strings 函数结束)返货参数和环境空间中已复制参数的头部偏移值	执行 return p
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.	
52	后重复 23-50 步复制脚本文件名	
53	后重复 23-50 步复制解释文件程序名	
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	
60	fs 指回用户段	复原 fs 的值
61	(跳到程序开始界面)返回前部分循环执行	使用 goto 语句返回 restart 节点
62	(接 13 步,不为脚本) (动画)释放 bh 缓冲块	调用 brelse(bh)释放 bh 块
63	(长判断动画)判断目标文件是否可执行	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
64	(若不能执行)(跳错误界面)置错误码,跳 error2	给 retval 复制,执行 goto 语句
65	(判断动画)判断环境变量页面是否已经复制	直接读取 sh_bang 中的内容
66	(若未复制)复制命令行参数和环境字符串	同上述过程,调用 copy_strings 分别复制命令行

		参数和环境字符串
67	(若无空间) (跳错误界面)置错误码,跳 error2	给 retval 复制,执行 goto 语句
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	(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)
69	逐一关闭文件,更改文件句柄	将 current 中存放的当前进程打开的文件关闭
70	根据当前进程的基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	调用 free_page_tables 释放页面
71	根据之前加载的字符串长度计算出栈指针位置	页面最高位减去字符串总长即为栈指针 p 的位置
72	(动画)在栈空间中创建环境和参数变量指针表	调用 create_tables
73	修改 current 各字段的值为新执行文件的信息	用 ex 中各字段的值以及 e_uid 和 e_gid 中的数据修改 current 中的数据
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	eip[0] = ex.a_entry;
75	栈指针替换为新程序的栈指针 p	eip[3] = p;
76	(错误界面 error2)放回 i 节点	iput(inode)
77	(错误界面 error1)释放存放参数和环境串的内存页面,返回出错码	先调用 free_page 释放页面,后 return retval

第二部分 数据提取 共 25 分

2. 提取方法： 10 分

我们打算使用 gdb 脚本，在需要停止的代码处设置断点，在执行功能中输入要 print 的变量值，之后通过运行 run_gdb_script_hd_console_redirect 这一 function，将断点处输出的内容输出到 gdb_output.txt 中。将我们需要的变量值输出之后，我们打算写一个 python 脚本对这些字符串进行处理，将这些字符串转换成 json 格式的文件，之后根据我们的 json 文件去执行我们每一个动作的动画。

2. 提取目标：15分

编号	位置	目的
1	fs/exec.c 194行	提取194行中判断语句中eip[1]的值
2	fs/namei.c 329行	提取此时返回的中dir这一inode结构体指针内存空间中内容
3	fs/exec.c 208行	提取i，用于提取当前文件的权限
4	fs/exec.c 209行	提取e_uid，用于之后判断用户是否有权限执行该文件
5	fs/exec.c 210行	提取e_gid，用于之后判断用户是否有权限执行该文件
6	fs/exec.c 216行	提取i，用于提取当前进程的权限
7	fs/exec.c 218行	echo “User does not have permission to execute the file”
8	fs/exec.c 237行	提取buf，查看脚本解释程序名
9	fs/exec.c 246行	提取清洗后的cp，查看清洗后的脚本解释程序名
10	fs/exec.c 260行	提取i_arg和i_name，查看解析后的程序文件名和参数
11	fs/exec.c 114行	提取new_fs,old_fs，查看两个段寄存器中的地址
12	fs/exec.c 123行	提取tmp，查看指向参数字符串的指针
13	fs/exec.c 128行	echo "Arg is too long!!!!"
14	fs/exec.c 140行	echo "Can't apply page!"
15	fs/exec.c 145行	提取pag，offset，查看当前进程地址
16	fs/exec.c 261行	提取sh_bang，判断脚本的参数和环境变量是否已经加载到内存
17	fs/exec.c 311行	提取sh_bang，判断是否需要加载命令行参数和环境字符串到内存
18	fs/exec.c 315行	echo "Stack is empty!!!"
19	fs/exec.c 322行	提取executable，监视当前进程是否可执行
20	fs/exec.c 201行	提取inode，用于与namei进行比较
21	fs/exec.c 225行	提取ex，用于演示读取文件头的过程
22	fs/exec.c 343行	提取current，用于展示当前进程信息，此时进程切换完毕
23	fs/exec.c 280行	用于判断当前p是否为0
24	需要多次展示fs和ds寄存器	主要将fs显示出来，查看当前进程基地址是否与数据段基地址相同。
25	fs/exec.c 297行与299行	提取bh缓冲块，观察其是否被释放
26	fs/exec.c 135行	提取from_kmem的值，用于之后判断其值
27	fs/exec.c 136行	提取offset和p的值，用于展示将p存入offset

注：此处使用的源代码以linux0.11lab中的代码为准。（如果使用其他代码，请修改此处）

第三部分 可视化 共 30 分

3. 界面： 20 分

图 1

图 2


当前进行的动作
（此处注明动作序号）
说明变量的值
说明变量的值
由于大部分过程为比较过程，
特此创建该比较结果框用于
展示结果

图 3


ERROR：（后面显示错误类型）

图 4

图 5

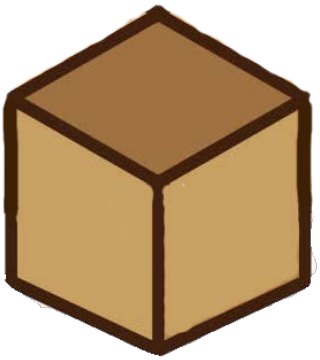

图 6

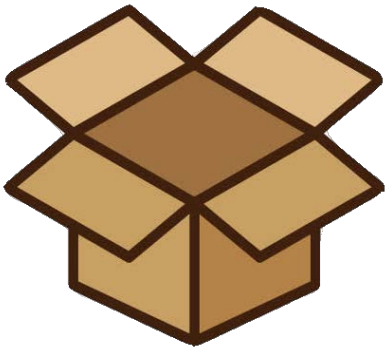

图 7

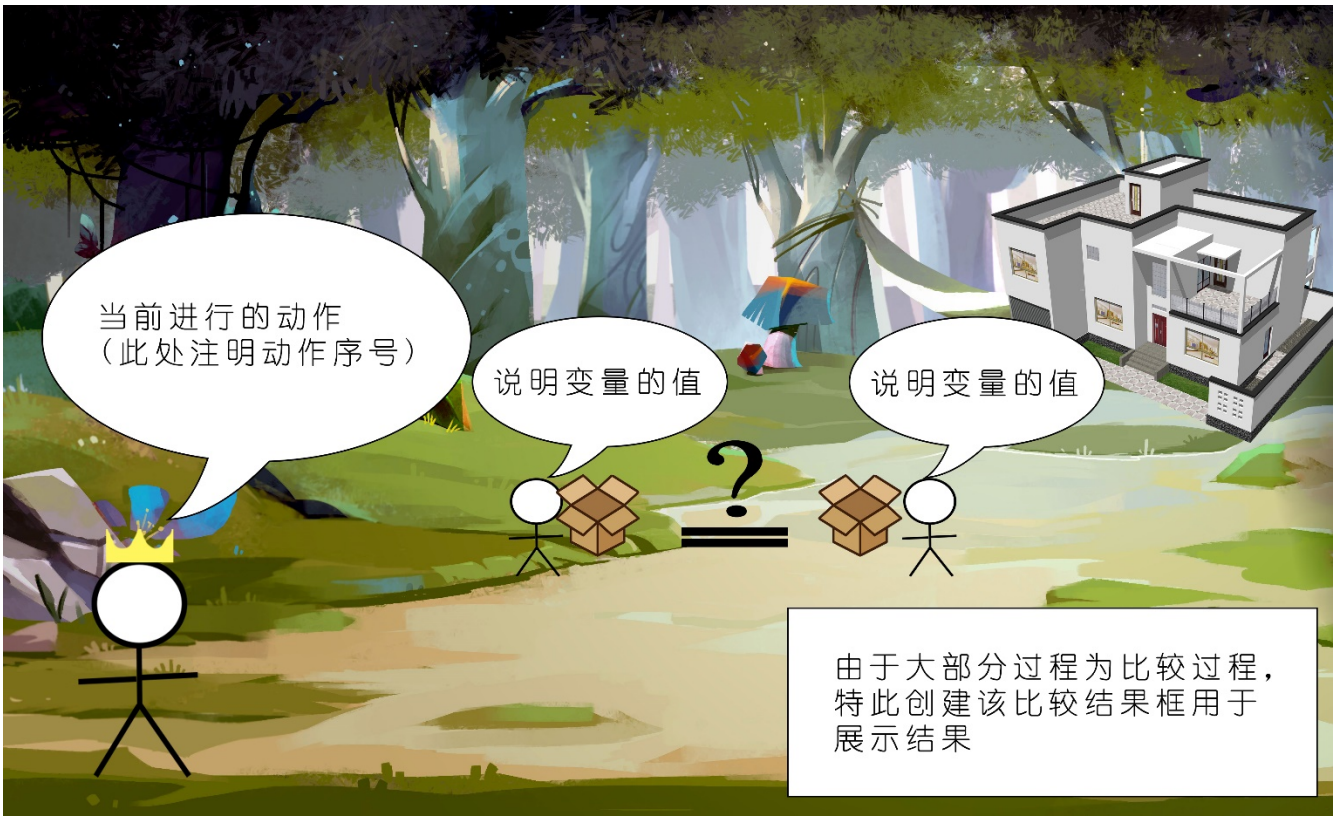

图 8 第一类动作展示界面

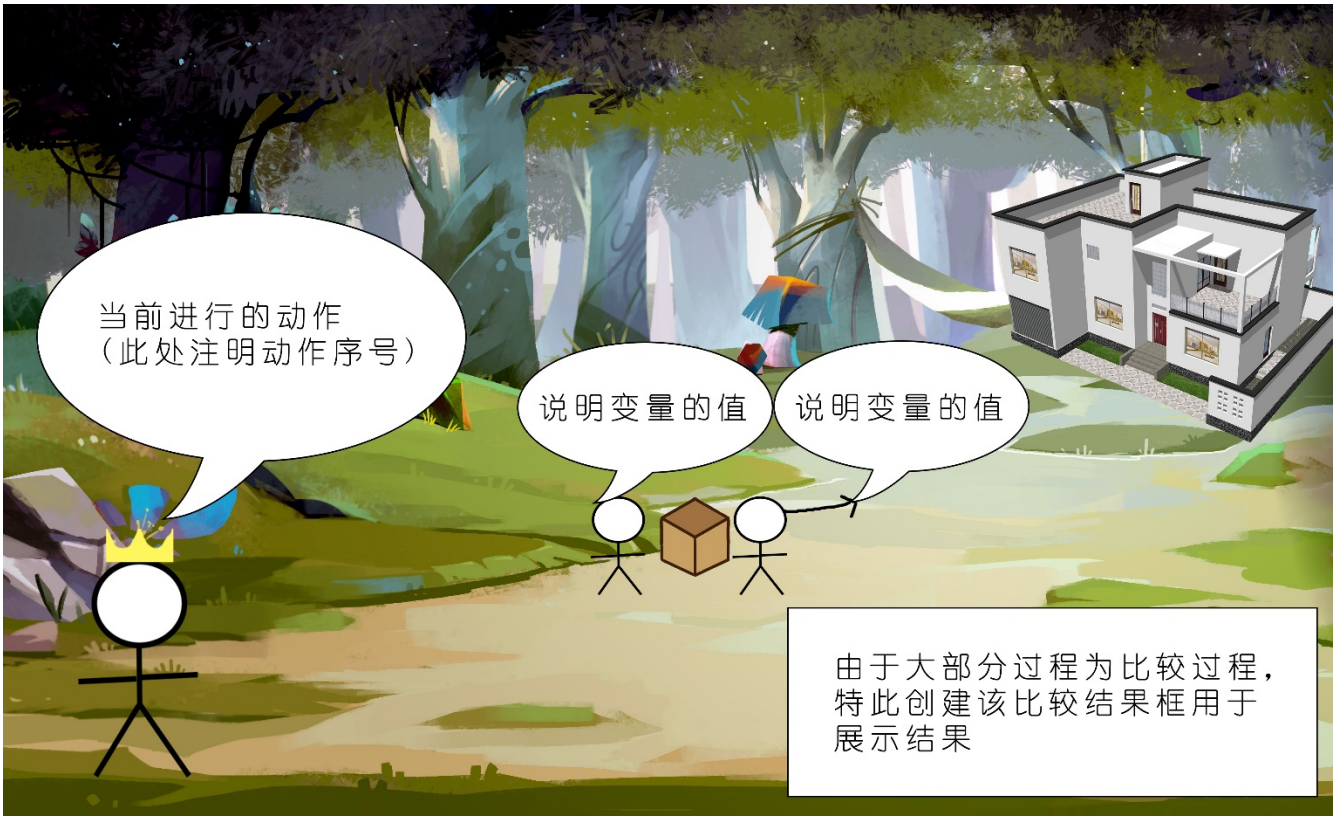

图 9 第二类动作展示界面

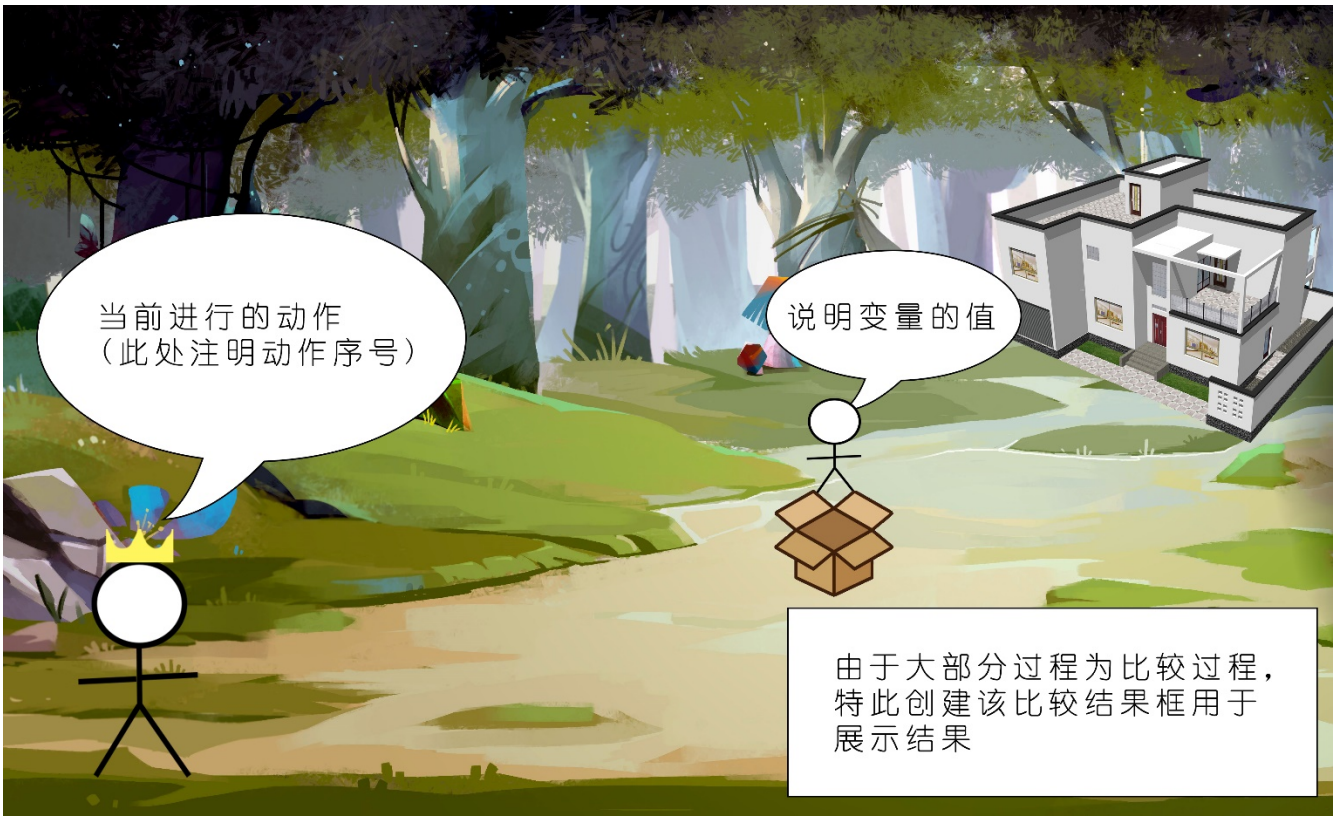

图 10 第三类动作展示界面

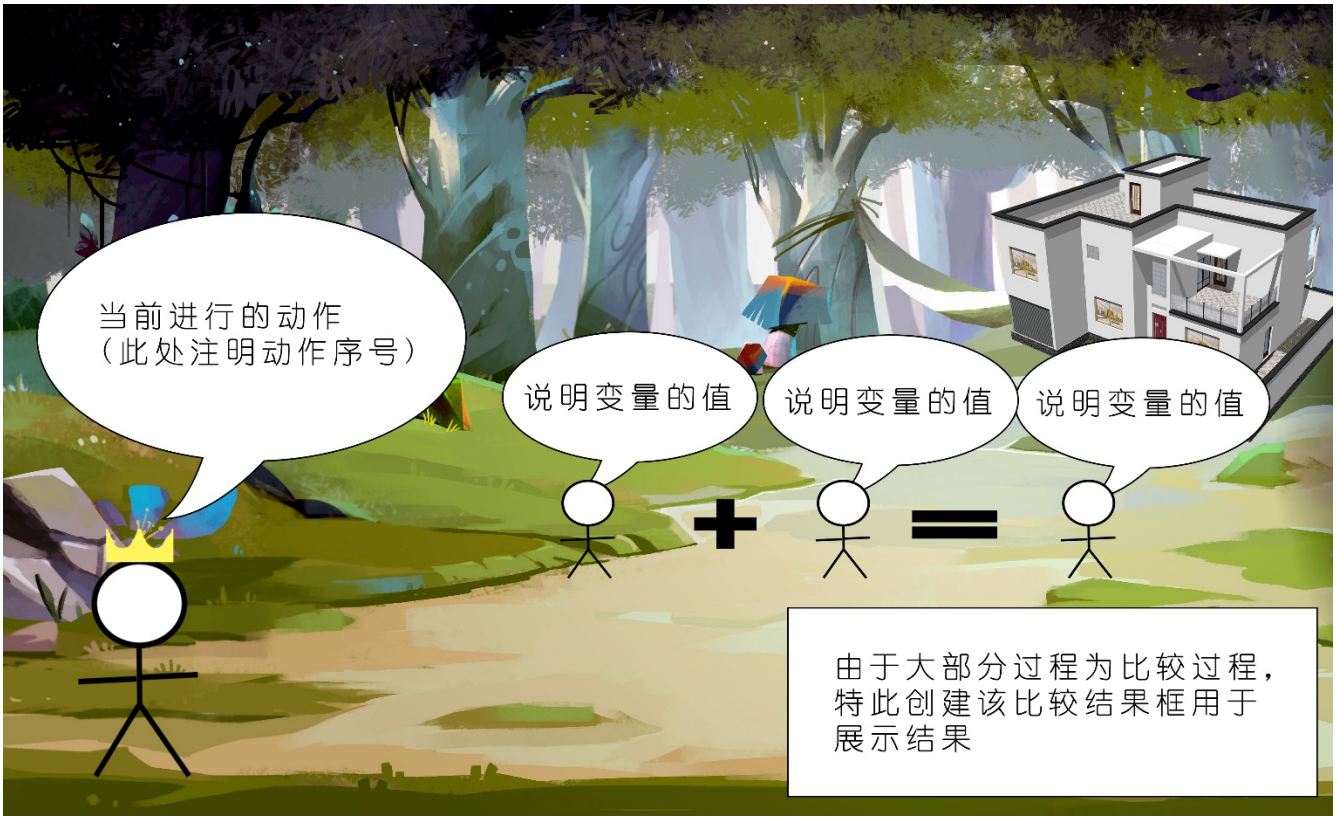

图 11 第五类动作展示界面

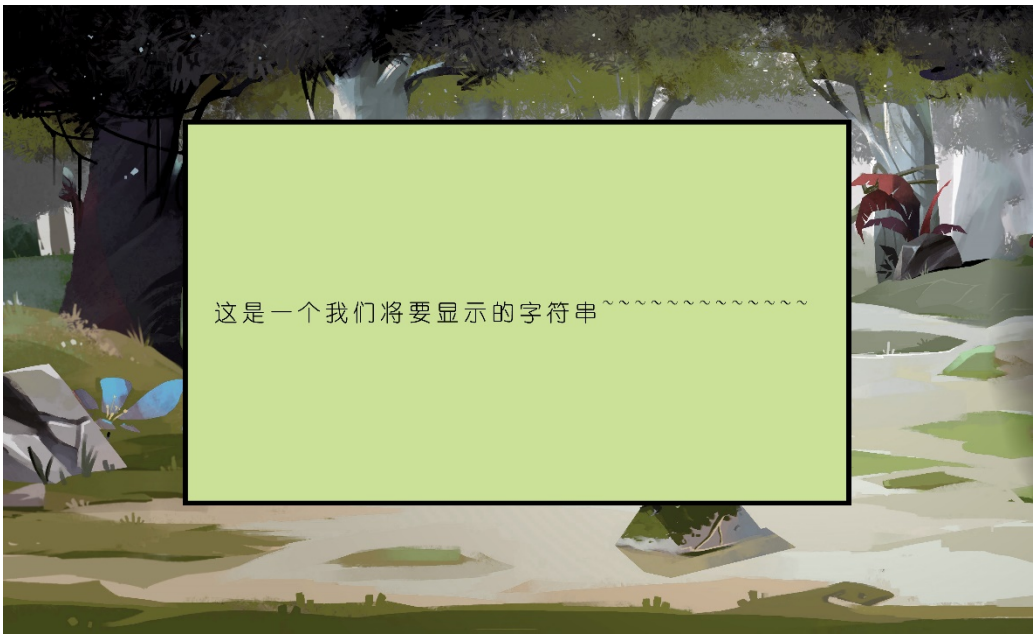

图 12 第六类动作展示界面

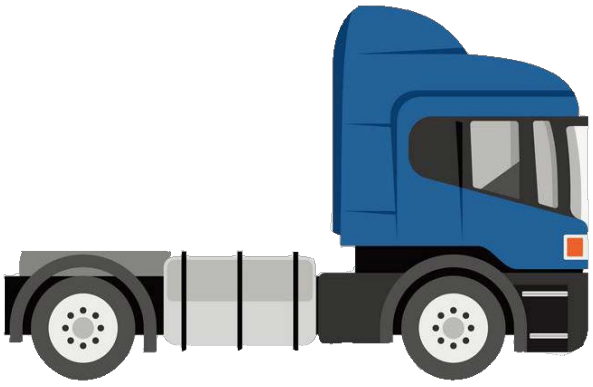

图 12 素材

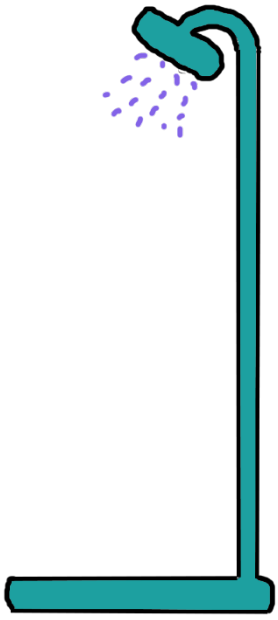

图 13 素材

上述三个页面分别为开始页面，do_execve()函数的执行页面，copy_string()函数的执行页面，具体可视化展示方式见图中细节。我们的动画过程大致分为以下几类：

1. 第一类为判断两个变量的值是否相等，该场景主要用于一些条件判断语句中，因此我们在场景右下角特别设置了文本框来记录这一过程判断的最终结果，以及最后要跳转的方向。在后面的描述中我们用（判断东华）来标记这一类动画，动画中展示方式为将两个小人（变量）同时拿出一个打开的盒子（如图 7）相对进行比较，最后展示效果如图 8。
2. 第二类为将一个变量的值赋值给另一个变量，该场景主要用于切换新进程进行执行时，需要将新进程的许多变量赋值给其他变量。我们计划使用将一个小人传递一个箱子（如图 6）给另一个小人的动作来表示这一过程，最后展示效果如图 9。
3. 第三类为提取数据变量的动画，该场景我们计划使用一个小人打开箱子呈现给大家的方式进行表示，最后展示结果如图 10。
4. 第四类为错误跳转动画，该场景我们计划使用一个特殊的页面（如图 4）进行展示，该场景中央的文本框会为大家展示错误信息。
5. 第五类为算术逻辑运算动画，当有变量进行算术逻辑运算时，为了展示清晰，我们在小人上方的对话框里面进行文本展示，最后展示结果如图 11。
6. 第六类为字符串显示动画，由于该步骤在之前已经重复过，或该步骤代表了某一状态的转变等情况，我们仅显示一个字符串来讲述当前发生的事情，最后展示结果如图 12。
7. 剩余不属于五大类的动作我们将在后面事件的动画展示中进行详细文字描述。特殊动画包括动画 1,15,23,42,43,51,66，最后可能的一些素材如图 13,14 所示。

3. 操作方法：5 分

操作者只需要选择执行的一套数据，之后的过程便会自行根据选择的数据开始进行演示。最后如果成功执行完毕将会跳转进入最后的结束界面（如图 5），此外我们还计划在动画中加入重置，暂停，以及倍速功能，具体实现方式待定。

3. 事件的动画表示：5 分

这一部分我们暂时采用文字进行描述，以后有机会我们会对其进行图片完善，文字描述如下：

1) 第一个动作为操作者选择进入哪一个故事中，两个故事将会采取两组不同的数据，第一组模拟执行 shell 脚本的情况，第二组模拟执行可执行文件的情况。
2) （第六类字符串显示动画）”故事开始（do_execve()开始执行）”。
3) （第三类读取动画）动画中读取变量 eip[1]的值以小人的方式进行呈现。
4) （第一类比较动画）动画中比较动作 4 中变量和常量。
5) （第三类读取动画）动画中直接读取函数 namei()调用的返回值。
6) （第一类判断动画）动画中比较动作 6 中两个变量。
7) （第一类读取动画）动画中直接读取 i_mode 数值。
8) （第五类算术逻辑运算动画）动画中对 e_uid 进行移位运算的展示。
9) （第五类算术逻辑运算动画）动画中对 e_gid 进行移位运算的展示。
10) （第五类算术逻辑运算动画）动画中对 i_mode 进行运算的展示。
11) （第四类错误跳转动画）动画进入错误界面。
12) （第三类读取动画）读取赋值后 ex 的值。

13) （第一类判断动画）动画中比较动作 13 中变量和常量。
14) （第三类读取动画）读取 buf 中存储的内容。
15) 该动画我们计划使用小人将箱子释放后箱子消失的方式进行展示。
16) （第二类赋值动画）将 inode 进行赋值，另一个赋值的小人用函数名称 iput()进行表示。
17) 该动画我们计划用一个清洗的特效进行展示，之后呈现一个第三类读取动画展示清洗后的内容。
18) （第一类判断动画）动画中比较 buf 中内容和空常量。
19) （第四类错误跳转动画）略。
20) （第三类读取动画）读取 i_name 和 i_arg 的值。
21) （第五类算术逻辑运算动画）将 sh_bang 的值加 1。
22) （第六类字符串显示动画）“Chapter 2（已经来到了 copy_string()函数）”。
23) 单独呈现场景 2。
24) （第二类赋值动画）get_ds()的值赋值给 new_fs。
25) （第二类赋值动画）get_fs()的值赋值给 old_fs。
26) （第一类判断动画）判断动作 26 中变量和常量。
27) （第二类赋值动画）对 fs 进行赋值，赋值变量名为 set_fs()。
28) （第六类字符串显示动画）“下面的内容将循环进行，无耐心请暂停后倍速播放。”。
29) （第一类判断动画）判断动作 29 中变量和常量。
30) （第二类赋值动画）同 27。
31) （第二类赋值动画）将 get_fs_long()赋值给 fs。
32) （第一类判断动画）同 29。
33) （第二类赋值动画）同 27。
34) （第三类读取动画）读取 get_fs_byte()。
35) （第一类判断动画）判断 len 和 0 之间关系。
36) （第四类错误跳转动画）略。
37) （第六类字符串显示动画）同 28。
38) （第五类算术逻辑运算动画）将运算后得到的 p 指针偏移值进行存放。
39) （第一类判断动画）同 29。
40) （第二类赋值动画）将 fs 进行赋值。
41) （第一类判断动画）将 p 所在空间与空（null）进行比较。
42) 显示正在执行 get_free_page()函数，可以添加一个卡车运输特效。
43) 直接返回场景 1，同时将程序内变量返回一个 0。
44) （第一类判断动画）同 26。
45) （第二类赋值动画）同 27
46) （第五类算术逻辑运算动画）将 pag 和 offset 相加。
47) （第二类赋值动画）读取 pag 与 offset 相加地址处的值。
48) （第一类判断动画）同 26。
49) （第二类赋值动画）同 40。
50) （第六类字符串显示动画）“Chapter 2 的故事告一段落。（copy_strings()函数执行结束）。”
51) 显示场景 1
52) （第六类字符串显示动画）“Chapter 2 将被永远记住。（copy_strings()函数复制脚本文件名）。”
53) （第六类字符串显示动画）“Chapter 2 将被永远记住。（copy_strings()函数复制解释文件程序名）。”
54) （第六类字符串显示动画）“Chapter 2 将被永远记住。（copy_strings()函数复制解释文件参数）。”
55) （第一类判断动画）判断程序内 p 值。

56) （第四类错误跳转动画）略。
57) （第二类赋值动画）对 fs 进行赋值。
58) （第三类读取动画）读取 i 结点。
59) （第四类错误跳转动画）略。
60) （第二类赋值动画）对 fs 进行赋值。（由于对 fs 寄存器频繁操作，该一段步骤我们也可以实时展示 fs 的值，具体实现方案待定）
61) （第六类字符串展示动画）“我们又回到了开始。（跳转到 restart 结点）”。
62) 同 15。
63) （第一类判断动画）判断文件是否可以执行，用 exe_able()表示。
64) （第四类错误跳转动画）略。
65) （第一类判断动画）将 sh_bang 中的值和 1 比较。
66) 返回动作 22。
67) （第四类错误跳转动画）
68) （第二类赋值动画）对 executable 进行赋值
69) （第二类赋值动画）对 current 进行赋值，关闭文件。
70) （第六类字符串显示动画）“让我们放轻松（调用 free_page_tables()释放页面）”
71) （第五类算术逻辑运算动画）用页面最高位减去字符串总长。
72) （第六类字符串显示动画）“让我们创造新世界（调用 create_tables()创建页面）”
73) （第二类赋值动画）对 e_uid，e_gid，current 赋值。
74) （第二类赋值动画）将 eip[0]进行赋值。
75) （第二类赋值动画）将 p 赋值给 eip[3]。
76) （第四类错误跳转动画）略。
77) （第四类错误跳转动画）略。

第四部分 实验报告计划 共 20 分

4. 提取数据部分： 10 分

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，我们计划写一个 python 脚本将每一条数据处理成一个 json 对象，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。

4. 可视化部分：10 分

我们已经在腾讯云平台上搭建了一个 web 服务器，我们计划用网页的方式将我们的动画进行呈现，使用语言主要有 HTML5，JavaScript，CSS3 以及 Jquery。我们主要展示的是 exec.c 文件中 do_execve()函数的执行过程，并且还对其调用本文件中的 copy_string()进行反复展示。因此我们计划设计两个场景进行主要展示，并且将这两个函数通过不同的网页切换来达到切换的目的，在具体展示的过程中，上述 json 对象中的备注内容也会被我们展示出来，这样我们能够更加方便的为大家呈现这个函数执行的具体细节。最后我们将把这些文件上传到 https://github.com/gxy666/OS_PRO 中的”web 可视化网站”文件夹中

详细分工：

本组组员荆嘉政，高翔宇。

荆嘉政负责事件细节的整理，之后负责数据提取，将 gdb_output.txt 中的数据编写脚本处理为 json 对象等工作，并在可视化过程中负责搜集和预处理素材，设计网页样式等辅助工作。

高翔宇主要负责可视化部分：将 json 格式的文件读入，然后模块化的编写上述各中动画场景并将其按照执行过程组合起来。此外还负责服务器搭建和调试等工作。