
系统综合课程设计-实验报告

PA3-穿越时空的旅程：异常控制流

姓名：孙铭

学号：1711377

学院：计算机学院

专业：计算机科学与技术

时间：2020年5月27日

目录

系统综合课程设计-实验报告

PA3-穿越时空的旅程：异常控制流

目录

PA3: 阶段一

1. 加载操作系统的第一个用户程序

1.1 准备工作

1.2 代码：实现loader

2. 穿越时空的旅程

2.1 准备IDT

2.2 触发异常

2.3 保存现场

思考题一：对比异常与函数调用

思考题二：诡异的代码

2.4 事件分发

2.5 系统调用处理

2.6 恢复现场

PA3: 阶段二

3.1 代码：在Nanos-lite上运行Hello world

3.2 堆区管理

3.3 简易文件系统

3.4 实现完整文件系统

PA3: 阶段三

4.1 代码：把VGA显存抽象成文件

4.2 代码：把设备输入抽象成文件

4.3 运行仙剑奇侠传

4.4 必答题

下面进入本次实验内容分析。

PA3: 阶段一

阶段一的内容主要是熟悉操作系统基本概念、系统调用以及实现中断机制。接下来逐步展示各个模块代码实现过程如下。

1. 加载操作系统的第一个用户程序

1.1 准备工作

首先，根据实验指导，需要解决四个问题：

- 1. 可执行文件在哪里？
- 2. 代码和数据在可执行文件的哪个位置？
- 3. 代码和数据有多少？
- 4. “正确的内存位置”在哪里？

尽管指导中给出了相关问题的答案，但这里还是先梳理一下。

问题一：可执行文件在哪里

需要让Navy-apps项目上的程序默认编译到x86中，在`navy-apps/Makefile.check`下修改编译环境如下。

```
1 ISA ?= x86
2 ifeq ($(NAVY_HOME), )
3   $(error Must set NAVY_HOME environment variable)
4 endif
5
6 $(shell mkdir -p $(NAVY_HOME)/fsimg/bin/ $(NAVY_HOME)/fsimg/dev/)
```

之后在`navy-apps/tests/dummy`下执行`make`命令，出现大量warning，根据实验指导中，这是正常现象。为了避免和nanos-lite内容产生冲突，我们约定目前用户程序需要被链接到内存位置0x4000000处。

接下来在`nanos-lite/`下执行`make update`指令，运行结果如下。

```
sun@ests:~/Desktop/ics2018/nanos-lite$ make update
Building nanos-lite [x86-nemu]
objcopy -S --set-section-flags .bss=alloc,contents -O binary /home/sun/Desktop/ics2018/navy-apps/tests/dummy/build/dummy-x86 build/ramdisk.img
touch src/files.h
ln -sf /home/sun/Desktop/ics2018/navy-apps/libs/libos/src/syscall.h src/syscall.h
```

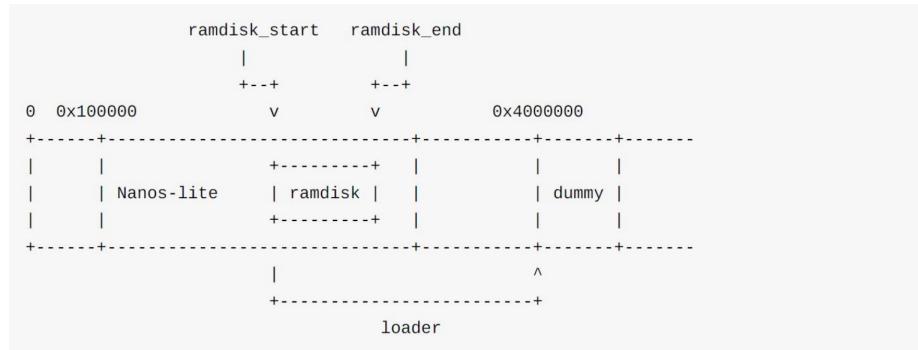
可以看到生成了ramdisk的镜像文件ramdisk.img，并包含进Nanos-lite成为其中的一部分。ramdisk非常简单，只包含了一个文件，即我们将要加载的用户程序。因此，第一个问题的回答就是：可执行文件位于ramdisk偏移为0处，访问它就可以得到用户程序的第一个字节。

问题二、三：代码和数据在可执行文件的哪个位置，分别有多少

参考实验手册，ELF文件格式除了包含程序本身的代码和静态数据之外，还包括一些用来描述它们的组织信息，loader目前并没有必要去解析并加载ELF文件，为了简化，`nanos-lite/Makefile`中已经把用户程序运行所需要的代码和静态数据通过object工具从ELF文件中抽取出了，整个ramdisk本身就已经存放了loader所需要加载的内容。

问题四：“正确的内存位置“在哪里

如前文所述正确的内存位置在代码中已经约定好，位于0x4000000。因此目前的loader只需做一件事，即将ramdisk中从0开始的所有内容放在0x4000000，并将这个地址作为程序的入口返回即可，通过内存布局理解loader目前需要做的事如下。



1.2 代码：实现loader

参照实验手册，任务要求如下。

实现 loader

你需要在Nanos-lite中实现loader的功能，来把用户程序加载到正确的内存位置，然后执行用户程序。需要注意的是，每当ramdisk中的内容需要更新时，你都需要在nanos-lite/目录下手动执行
make update
来更新Nanos-lite中的ramdisk内容，然后再通过
make run
来在NEMU上运行带有最新版ramdisk的Nanos-lite。
实现正确后，你会看到dummy程序执行了一条未实现的int指令，这说明loader已经成功加载dummy，并且成功地跳转到dummy中执行了。未实现的int指令我们会接下来的内容中进行说明。

目前，loader只需将ramdisk的从0开始的内容放置在0x4000000处，并返回该地址。使用extern添加外部变量与函数定义，调用ramdisk_read进行ramdisk的读取。在文件`nanos-lite/src/loader.c`中添加如下代码。

```
#include "common.h"

#define DEFAULT_ENTRY ((void *)0x4000000)

// PA3-1
extern uint8_t ramdisk_start;
extern uint8_t ramdisk_end;
#define RAMDISK_SIZE ((&ramdisk_end) - (&ramdisk_start))
extern void ramdisk_read(void *buf, off_t offset, size_t len);

uintptr_t loader(_Protect *as, const char *filename) {
    // TODO();
    ramdisk_read(DEFAULT_ENTRY, 0, RAMDISK_SIZE);
    return (uintptr_t)DEFAULT_ENTRY;
}
```

在`nanos-lite`下执行`make run`，运行结果如下。

```
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 10:17:16, May 29 2020
For help, type "help"
(nemu) c
[!]src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 17:19:51, May 28 2020
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x100cf0, end = 0x1052cc,
size = 17884 bytes
invalid opcode(eip = 0x04001f98): cd 80 5b 5d c3 66 90 90 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x04001f98 is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x04001f98) in the disassembling result to distinguish which case
it is.
```

在`navy-apps/tests/dummy/build`下使用命令`objdump -S dummy-x86`，可以查看可执行文件dummy-x86的反汇编代码，其中eip=0x4001f98处为未实现的int指令。

```

sun@ests: ~/Desktop/ics2018/navy-apps/tests/dummy/build
4001f86:    5d          pop   %ebp
4001f87:    c3          ret

04001f88 <_syscall_>:
4001f88:    55          push  %ebp
4001f89:    89 e5        mov    %esp,%ebp
4001f8b:    53          push  %ebx
4001f8c:    8b 55 14     mov    0x14(%ebp),%edx
4001f8f:    8b 4d 10     mov    0x10(%ebp),%ecx
4001f92:    8b 45 08     mov    0x8(%ebp),%eax
4001f95:    8b 5d 0c     mov    0xc(%ebp),%ebx
4001f98:    cd 80        int   $0x80
4001f9a:    5b          pop   %ebx
4001f9b:    5d          pop   %ebp
4001f9c:    c3          ret
4001f9d:    66 90        xchg  %ax,%ax
4001f9f:    90          nop

04001fa0 <_exit>:
4001fa0:    55          push  %ebp
4001fa1:    89 e5        mov    %esp,%ebp
4001fa3:    53          push  %ebx
4001fa4:    31 c9        xor   %ecx,%ecx
4001fa6:    b8 04 00 00 00  mov    $0x4,%eax

```

2. 穿越时空的旅程

2.1 准备IDT

首先，需要添加IDTR寄存器。

在目录 `nemu/include/cpu/reg.h` 中添加如下代码。

```

49      // 实现IDTR寄存器
50      struct IDTR{
51          uint32_t base;  // 首地址
52          uint16_t limit; // 长度
53      } idtr;

```

其次，需要实现lidt指令。

在 `nemu/include/cpu/decode.h` 中声明 `lidt_a` 的译码函数如下。

```
115 make_DHelper(lidt_a);
```

在 `nemu/src/cpu/decode/decode.c` 中实现 `lidt_a` 的译码函数如下。

```

312 make_DHelper(lidt_a){
313     decode_op_a(eip, id_dest, true);
314 }

```

接下来需要实现lidt的执行函数。

在 `nemu/src/cpu/exec/all-instr.h` 中声明执行函数如下。

```
56 make_EHelper(lidt);
```

在 `nemu/src/cpu/exec/system.c` 中实现执行函数如下。

```

6 make_EHelper(lidt) {
7     //TODO();
8
9     t1 = id_dest->val;
10    rtl_lm(&t0, &t1, 2);
11    cpu.idtr.limit = t0;
12
13    t1 = id_dest->val + 2;
14    rtl_lm(&t0, &t1, 4);
15    cpu.idtr.base = t0;
16
17 #ifdef DEBUG
18     Log("idtr.limit=0x%x", cpu.idtr.limit);
19     Log("idtr.base=0x%x", cpu.idtr.base);
20 #endif
21
22     print_asm_template1(lidt);
23 }

```

在 `nemu/src/exec/exec.c` 目录下添加 `opcode_table` 中注册指令如下。

```

67 /* 0x0f 0x01 */
68 make_group(gp7,
69     EMPTY, EMPTY, EMPTY, IDEX(lidt_a, lidt),
70     EMPTY, EMPTY, EMPTY, EMPTY)
71
72 /* 0x00 */
73     EMPTY, IDEX(gp7_E, gp7), EMPTY, EMPTY,
74
75 /* 0x01 */
76     EMPTY, EMPTY, EMPTY, EMPTY)
77
78 /* 0x02 */
79     EMPTY, EMPTY, EMPTY, EMPTY)
80
81 /* 0x03 */
82     EMPTY, EMPTY, EMPTY, EMPTY)
83
84 /* 0x04 */
85     EMPTY, EMPTY, EMPTY, EMPTY)
86
87 /* 0x05 */
88     EMPTY, EMPTY, EMPTY, EMPTY)
89
90 /* 0x06 */
91     EMPTY, EMPTY, EMPTY, EMPTY)
92
93 /* 0x07 */
94     EMPTY, EMPTY, EMPTY, EMPTY)
95
96 /* 0x08 */
97     EMPTY, EMPTY, EMPTY, EMPTY)
98
99 /* 0x09 */
100    EMPTY, EMPTY, EMPTY, EMPTY)
101
102 /* 0x0a */
103    EMPTY, EMPTY, EMPTY, EMPTY)
104
105 /* 0x0b */
106    EMPTY, EMPTY, EMPTY, EMPTY)
107
108 /* 0x0c */
109    EMPTY, EMPTY, EMPTY, EMPTY)
110
111 /* 0x0d */
112    EMPTY, EMPTY, EMPTY, EMPTY)
113
114 /* 0x0e */
115    EMPTY, EMPTY, EMPTY, EMPTY)
116
117 /* 0x0f */
118    EMPTY, EMPTY, EMPTY, EMPTY)
119
120 /* 0x10 */
121    EMPTY, EMPTY, EMPTY, EMPTY)
122
123 /* 0x11 */
124    EMPTY, EMPTY, EMPTY, EMPTY)
125
126 /* 0x12 */
127    EMPTY, EMPTY, EMPTY, EMPTY)
128
129 /* 0x13 */
130    EMPTY, EMPTY, EMPTY, EMPTY)
131
132 /* 0x14 */
133    EMPTY, EMPTY, EMPTY, EMPTY)
134
135 /* 0x15 */
136    EMPTY, EMPTY, EMPTY, EMPTY)
137
138 /* 0x16 */
139    EMPTY, EMPTY, EMPTY, EMPTY)
140
141 /* 0x17 */
142    EMPTY, EMPTY, EMPTY, EMPTY)
143
144 /* 0x18 */
145    EMPTY, EMPTY, EMPTY, EMPTY)
146
147 /* 0x19 */
148    EMPTY, EMPTY, EMPTY, EMPTY)
149
150 /* 0x1a */
151    EMPTY, EMPTY, EMPTY, EMPTY)
152
153 /* 0x1b */
154    EMPTY, EMPTY, EMPTY, EMPTY)
155
156 /* 0x1c */
157    EMPTY, EMPTY, EMPTY, EMPTY)
158
159 /* 0x1d */
160    EMPTY, EMPTY, EMPTY, EMPTY)
161
162 /* 0x1e */
163    EMPTY, EMPTY, EMPTY, EMPTY)
164
165 /* 0x1f */
166    EMPTY, EMPTY, EMPTY, EMPTY)
167
168 /* 0x20 */
169    EMPTY, EMPTY, EMPTY, EMPTY)
170
171 /* 0x21 */
172    EMPTY, EMPTY, EMPTY, EMPTY)
173
174 /* 0x22 */
175    EMPTY, EMPTY, EMPTY, EMPTY)
176
177 /* 0x23 */
178    EMPTY, EMPTY, EMPTY, EMPTY)
179
180 /* 0x24 */
181    EMPTY, EMPTY, EMPTY, EMPTY)
182
183 /* 0x25 */
184    EMPTY, EMPTY, EMPTY, EMPTY)
185
186 /* 0x26 */
187    EMPTY, EMPTY, EMPTY, EMPTY)
188
189 /* 0x27 */
190    EMPTY, EMPTY, EMPTY, EMPTY)
191
192 /* 0x28 */
193    EMPTY, EMPTY, EMPTY, EMPTY)
194
195 /* 0x29 */
196    EMPTY, EMPTY, EMPTY, EMPTY)
197
198 /* 0x2a */
199    EMPTY, EMPTY, EMPTY, EMPTY)
200
201 /* 0x2b */
202    EMPTY, EMPTY, EMPTY, EMPTY)
203
204 /* 0x2c */
205    EMPTY, EMPTY, EMPTY, EMPTY)
206
207 /* 0x2d */
208    EMPTY, EMPTY, EMPTY, EMPTY)
209
210 /* 0x2e */
211    EMPTY, EMPTY, EMPTY, EMPTY)
212
213 /* 0x2f */
214    EMPTY, EMPTY, EMPTY, EMPTY)
215
216 /* 0x30 */
217    EMPTY, EMPTY, EMPTY, EMPTY)
218
219 /* 0x31 */
220    EMPTY, EMPTY, EMPTY, EMPTY)
221
222 /* 0x32 */
223    EMPTY, EMPTY, EMPTY, EMPTY)
224
225 /* 0x33 */
226    EMPTY, EMPTY, EMPTY, EMPTY)
227
228 /* 0x34 */
229    EMPTY, EMPTY, EMPTY, EMPTY)
230
231 /* 0x35 */
232    EMPTY, EMPTY, EMPTY, EMPTY)
233
234 /* 0x36 */
235    EMPTY, EMPTY, EMPTY, EMPTY)
236
237 /* 0x37 */
238    EMPTY, EMPTY, EMPTY, EMPTY)
239
240 /* 0x38 */
241    EMPTY, EMPTY, EMPTY, EMPTY)
242
243 /* 0x39 */
244    EMPTY, EMPTY, EMPTY, EMPTY)
245
246 /* 0x3a */
247    EMPTY, EMPTY, EMPTY, EMPTY)
248
249 /* 0x3b */
250    EMPTY, EMPTY, EMPTY, EMPTY)
251
252 /* 0x3c */
253    EMPTY, EMPTY, EMPTY, EMPTY)
254
255 /* 0x3d */
256    EMPTY, EMPTY, EMPTY, EMPTY)
257
258 /* 0x3e */
259    EMPTY, EMPTY, EMPTY, EMPTY)
260
261 /* 0x3f */
262    EMPTY, EMPTY, EMPTY, EMPTY)
263
264 /* 0x40 */
265    EMPTY, EMPTY, EMPTY, EMPTY)
266
267 /* 0x41 */
268    EMPTY, EMPTY, EMPTY, EMPTY)
269
270 /* 0x42 */
271    EMPTY, EMPTY, EMPTY, EMPTY)
272
273 /* 0x43 */
274    EMPTY, EMPTY, EMPTY, EMPTY)
275
276 /* 0x44 */
277    EMPTY, EMPTY, EMPTY, EMPTY)
278
279 /* 0x45 */
280    EMPTY, EMPTY, EMPTY, EMPTY)
281
282 /* 0x46 */
283    EMPTY, EMPTY, EMPTY, EMPTY)
284
285 /* 0x47 */
286    EMPTY, EMPTY, EMPTY, EMPTY)
287
288 /* 0x48 */
289    EMPTY, EMPTY, EMPTY, EMPTY)
290
291 /* 0x49 */
292    EMPTY, EMPTY, EMPTY, EMPTY)
293
294 /* 0x4a */
295    EMPTY, EMPTY, EMPTY, EMPTY)
296
297 /* 0x4b */
298    EMPTY, EMPTY, EMPTY, EMPTY)
299
300 /* 0x4c */
301    EMPTY, EMPTY, EMPTY, EMPTY)
302
303 /* 0x4d */
304    EMPTY, EMPTY, EMPTY, EMPTY)
305
306 /* 0x4e */
307    EMPTY, EMPTY, EMPTY, EMPTY)
308
309 /* 0x4f */
310    EMPTY, EMPTY, EMPTY, EMPTY)
311
312 /* 0x50 */
313    EMPTY, EMPTY, EMPTY, EMPTY)
314
315 /* 0x51 */
316    EMPTY, EMPTY, EMPTY, EMPTY)
317
318 /* 0x52 */
319    EMPTY, EMPTY, EMPTY, EMPTY)
320
321 /* 0x53 */
322    EMPTY, EMPTY, EMPTY, EMPTY)
323
324 /* 0x54 */
325    EMPTY, EMPTY, EMPTY, EMPTY)
326
327 /* 0x55 */
328    EMPTY, EMPTY, EMPTY, EMPTY)
329
330 /* 0x56 */
331    EMPTY, EMPTY, EMPTY, EMPTY)
332
333 /* 0x57 */
334    EMPTY, EMPTY, EMPTY, EMPTY)
335
336 /* 0x58 */
337    EMPTY, EMPTY, EMPTY, EMPTY)
338
339 /* 0x59 */
340    EMPTY, EMPTY, EMPTY, EMPTY)
341
342 /* 0x5a */
343    EMPTY, EMPTY, EMPTY, EMPTY)
344
345 /* 0x5b */
346    EMPTY, EMPTY, EMPTY, EMPTY)
347
348 /* 0x5c */
349    EMPTY, EMPTY, EMPTY, EMPTY)
350
351 /* 0x5d */
352    EMPTY, EMPTY, EMPTY, EMPTY)
353
354 /* 0x5e */
355    EMPTY, EMPTY, EMPTY, EMPTY)
356
357 /* 0x5f */
358    EMPTY, EMPTY, EMPTY, EMPTY)
359
360 /* 0x60 */
361    EMPTY, EMPTY, EMPTY, EMPTY)
362
363 /* 0x61 */
364    EMPTY, EMPTY, EMPTY, EMPTY)
365
366 /* 0x62 */
367    EMPTY, EMPTY, EMPTY, EMPTY)
368
369 /* 0x63 */
370    EMPTY, EMPTY, EMPTY, EMPTY)
371
372 /* 0x64 */
373    EMPTY, EMPTY, EMPTY, EMPTY)
374
375 /* 0x65 */
376    EMPTY, EMPTY, EMPTY, EMPTY)
377
378 /* 0x66 */
379    EMPTY, EMPTY, EMPTY, EMPTY)
380
381 /* 0x67 */
382    EMPTY, EMPTY, EMPTY, EMPTY)
383
384 /* 0x68 */
385    EMPTY, EMPTY, EMPTY, EMPTY)
386
387 /* 0x69 */
388    EMPTY, EMPTY, EMPTY, EMPTY)
389
390 /* 0x6a */
391    EMPTY, EMPTY, EMPTY, EMPTY)
392
393 /* 0x6b */
394    EMPTY, EMPTY, EMPTY, EMPTY)
395
396 /* 0x6c */
397    EMPTY, EMPTY, EMPTY, EMPTY)
398
399 /* 0x6d */
400    EMPTY, EMPTY, EMPTY, EMPTY)
401
402 /* 0x6e */
403    EMPTY, EMPTY, EMPTY, EMPTY)
404
405 /* 0x6f */
406    EMPTY, EMPTY, EMPTY, EMPTY)
407
408 /* 0x70 */
409    EMPTY, EMPTY, EMPTY, EMPTY)
410
411 /* 0x71 */
412    EMPTY, EMPTY, EMPTY, EMPTY)
413
414 /* 0x72 */
415    EMPTY, EMPTY, EMPTY, EMPTY)
416
417 /* 0x73 */
418    EMPTY, EMPTY, EMPTY, EMPTY)
419
420 /* 0x74 */
421    EMPTY, EMPTY, EMPTY, EMPTY)
422
423 /* 0x75 */
424    EMPTY, EMPTY, EMPTY, EMPTY)
425
426 /* 0x76 */
427    EMPTY, EMPTY, EMPTY, EMPTY)
428
429 /* 0x77 */
430    EMPTY, EMPTY, EMPTY, EMPTY)
431
432 /* 0x78 */
433    EMPTY, EMPTY, EMPTY, EMPTY)
434
435 /* 0x79 */
436    EMPTY, EMPTY, EMPTY, EMPTY)
437
438 /* 0x7a */
439    EMPTY, EMPTY, EMPTY, EMPTY)
440
441 /* 0x7b */
442    EMPTY, EMPTY, EMPTY, EMPTY)
443
444 /* 0x7c */
445    EMPTY, EMPTY, EMPTY, EMPTY)
446
447 /* 0x7d */
448    EMPTY, EMPTY, EMPTY, EMPTY)
449
450 /* 0x7e */
451    EMPTY, EMPTY, EMPTY, EMPTY)
452
453 /* 0x7f */
454    EMPTY, EMPTY, EMPTY, EMPTY)
455
456 /* 0x80 */
457    EMPTY, EMPTY, EMPTY, EMPTY)
458
459 /* 0x81 */
460    EMPTY, EMPTY, EMPTY, EMPTY)
461
462 /* 0x82 */
463    EMPTY, EMPTY, EMPTY, EMPTY)
464
465 /* 0x83 */
466    EMPTY, EMPTY, EMPTY, EMPTY)
467
468 /* 0x84 */
469    EMPTY, EMPTY, EMPTY, EMPTY)
470
471 /* 0x85 */
472    EMPTY, EMPTY, EMPTY, EMPTY)
473
474 /* 0x86 */
475    EMPTY, EMPTY, EMPTY, EMPTY)
476
477 /* 0x87 */
478    EMPTY, EMPTY, EMPTY, EMPTY)
479
480 /* 0x88 */
481    EMPTY, EMPTY, EMPTY, EMPTY)
482
483 /* 0x89 */
484    EMPTY, EMPTY, EMPTY, EMPTY)
485
486 /* 0x8a */
487    EMPTY, EMPTY, EMPTY, EMPTY)
488
489 /* 0x8b */
490    EMPTY, EMPTY, EMPTY, EMPTY)
491
492 /* 0x8c */
493    EMPTY, EMPTY, EMPTY, EMPTY)
494
495 /* 0x8d */
496    EMPTY, EMPTY, EMPTY, EMPTY)
497
498 /* 0x8e */
499    EMPTY, EMPTY, EMPTY, EMPTY)
500
501 /* 0x8f */
502    EMPTY, EMPTY, EMPTY, EMPTY)
503
504 /* 0x90 */
505    EMPTY, EMPTY, EMPTY, EMPTY)
506
507 /* 0x91 */
508    EMPTY, EMPTY, EMPTY, EMPTY)
509
510 /* 0x92 */
511    EMPTY, EMPTY, EMPTY, EMPTY)
512
513 /* 0x93 */
514    EMPTY, EMPTY, EMPTY, EMPTY)
515
516 /* 0x94 */
517    EMPTY, EMPTY, EMPTY, EMPTY)
518
519 /* 0x95 */
520    EMPTY, EMPTY, EMPTY, EMPTY)
521
522 /* 0x96 */
523    EMPTY, EMPTY, EMPTY, EMPTY)
524
525 /* 0x97 */
526    EMPTY, EMPTY, EMPTY, EMPTY)
527
528 /* 0x98 */
529    EMPTY, EMPTY, EMPTY, EMPTY)
530
531 /* 0x99 */
532    EMPTY, EMPTY, EMPTY, EMPTY)
533
534 /* 0x9a */
535    EMPTY, EMPTY, EMPTY, EMPTY)
536
537 /* 0x9b */
538    EMPTY, EMPTY, EMPTY, EMPTY)
539
540 /* 0x9c */
541    EMPTY, EMPTY, EMPTY, EMPTY)
542
543 /* 0x9d */
544    EMPTY, EMPTY, EMPTY, EMPTY)
545
546 /* 0x9e */
547    EMPTY, EMPTY, EMPTY, EMPTY)
548
549 /* 0x9f */
550    EMPTY, EMPTY, EMPTY, EMPTY)
551
552 /* 0xa0 */
553    EMPTY, EMPTY, EMPTY, EMPTY)
554
555 /* 0xa1 */
556    EMPTY, EMPTY, EMPTY, EMPTY)
557
558 /* 0xa2 */
559    EMPTY, EMPTY, EMPTY, EMPTY)
560
561 /* 0xa3 */
562    EMPTY, EMPTY, EMPTY, EMPTY)
563
564 /* 0xa4 */
565    EMPTY, EMPTY, EMPTY, EMPTY)
566
567 /* 0xa5 */
568    EMPTY, EMPTY, EMPTY, EMPTY)
569
570 /* 0xa6 */
571    EMPTY, EMPTY, EMPTY, EMPTY)
572
573 /* 0xa7 */
574    EMPTY, EMPTY, EMPTY, EMPTY)
575
576 /* 0xa8 */
577    EMPTY, EMPTY, EMPTY, EMPTY)
578
579 /* 0xa9 */
580    EMPTY, EMPTY, EMPTY, EMPTY)
581
582 /* 0xa0 */
583    EMPTY, EMPTY, EMPTY, EMPTY)
584
585 /* 0xa1 */
586    EMPTY, EMPTY, EMPTY, EMPTY)
587
588 /* 0xa2 */
589    EMPTY, EMPTY, EMPTY, EMPTY)
590
591 /* 0xa3 */
592    EMPTY, EMPTY, EMPTY, EMPTY)
593
594 /* 0xa4 */
595    EMPTY, EMPTY, EMPTY, EMPTY)
596
597 /* 0xa5 */
598    EMPTY, EMPTY, EMPTY, EMPTY)
599
600 /* 0xa6 */
601    EMPTY, EMPTY, EMPTY, EMPTY)
602
603 /* 0xa7 */
604    EMPTY, EMPTY, EMPTY, EMPTY)
605
606 /* 0xa8 */
607    EMPTY, EMPTY, EMPTY, EMPTY)
608
609 /* 0xa9 */
610    EMPTY, EMPTY, EMPTY, EMPTY)
611
612 /* 0xa0 */
613    EMPTY, EMPTY, EMPTY, EMPTY)
614
615 /* 0xa1 */
616    EMPTY, EMPTY, EMPTY, EMPTY)
617
618 /* 0xa2 */
619    EMPTY, EMPTY, EMPTY, EMPTY)
620
621 /* 0xa3 */
622    EMPTY, EMPTY, EMPTY, EMPTY)
623
624 /* 0xa4 */
625    EMPTY, EMPTY, EMPTY, EMPTY)
626
627 /* 0xa5 */
628    EMPTY, EMPTY, EMPTY, EMPTY)
629
630 /* 0xa6 */
631    EMPTY, EMPTY, EMPTY, EMPTY)
632
633 /* 0xa7 */
634    EMPTY, EMPTY, EMPTY, EMPTY)
635
636 /* 0xa8 */
637    EMPTY, EMPTY, EMPTY, EMPTY)
638
639 /* 0xa9 */
640    EMPTY, EMPTY, EMPTY, EMPTY)
641
642 /* 0xa0 */
643    EMPTY, EMPTY, EMPTY, EMPTY)
644
645 /* 0xa1 */
646    EMPTY, EMPTY, EMPTY, EMPTY)
647
648 /* 0xa2 */
649    EMPTY, EMPTY, EMPTY, EMPTY)
650
651 /* 0xa3 */
652    EMPTY, EMPTY, EMPTY, EMPTY)
653
654 /* 0xa4 */
655    EMPTY, EMPTY, EMPTY, EMPTY)
656
657 /* 0xa5 */
658    EMPTY, EMPTY, EMPTY, EMPTY)
659
660 /* 0xa6 */
661    EMPTY, EMPTY, EMPTY, EMPTY)
662
663 /* 0xa7 */
664    EMPTY, EMPTY, EMPTY, EMPTY)
665
666 /* 0xa8 */
667    EMPTY, EMPTY, EMPTY, EMPTY)
668
669 /* 0xa9 */
670    EMPTY, EMPTY, EMPTY, EMPTY)
671
672 /* 0xa0 */
673    EMPTY, EMPTY, EMPTY, EMPTY)
674
675 /* 0xa1 */
676    EMPTY, EMPTY, EMPTY, EMPTY)
677
678 /* 0xa2 */
679    EMPTY, EMPTY, EMPTY, EMPTY)
680
681 /* 0xa3 */
682    EMPTY, EMPTY, EMPTY, EMPTY)
683
684 /* 0xa4 */
685    EMPTY, EMPTY, EMPTY, EMPTY)
686
687 /* 0xa5 */
688    EMPTY, EMPTY, EMPTY, EMPTY)
689
690 /* 0xa6 */
691    EMPTY, EMPTY, EMPTY, EMPTY)
692
693 /* 0xa7 */
694    EMPTY, EMPTY, EMPTY, EMPTY)
695
696 /* 0xa8 */
697    EMPTY, EMPTY, EMPTY, EMPTY)
698
699 /* 0xa9 */
700    EMPTY, EMPTY, EMPTY, EMPTY)
701
702 /* 0xa0 */
703    EMPTY, EMPTY, EMPTY, EMPTY)
704
705 /* 0xa1 */
706    EMPTY, EMPTY, EMPTY, EMPTY)
707
708 /* 0xa2 */
709    EMPTY, EMPTY, EMPTY, EMPTY)
710
711 /* 0xa3 */
712    EMPTY, EMPTY, EMPTY, EMPTY)
713
714 /* 0xa4 */
715    EMPTY, EMPTY, EMPTY, EMPTY)
716
717 /* 0xa5 */
718    EMPTY, EMPTY, EMPTY, EMPTY)
719
720 /* 0xa6 */
721    EMPTY, EMPTY, EMPTY, EMPTY)
722
723 /* 0xa7 */
724    EMPTY, EMPTY, EMPTY, EMPTY)
725
726 /* 0xa8 */
727    EMPTY, EMPTY, EMPTY, EMPTY)
728
729 /* 0xa9 */
730    EMPTY, EMPTY, EMPTY, EMPTY)
731
732 /* 0xa0 */
733    EMPTY, EMPTY, EMPTY, EMPTY)
734
735 /* 0xa1 */
736    EMPTY, EMPTY, EMPTY, EMPTY)
737
738 /* 0xa2 */
739    EMPTY, EMPTY, EMPTY, EMPTY)
740
741 /* 0xa3 */
742    EMPTY, EMPTY, EMPTY, EMPTY)
743
744 /* 0xa4 */
745    EMPTY, EMPTY, EMPTY, EMPTY)
746
747 /* 0xa5 */
748    EMPTY, EMPTY, EMPTY, EMPTY)
749
750 /* 0xa6 */
751    EMPTY, EMPTY, EMPTY, EMPTY)
752
753 /* 0xa7 */
754    EMPTY, EMPTY, EMPTY, EMPTY)
755
756 /* 0xa8 */
757    EMPTY, EMPTY, EMPTY, EMPTY)
758
759 /* 0xa9 */
760    EMPTY, EMPTY, EMPTY, EMPTY)
761
762 /* 0xa0 */
763    EMPTY, EMPTY, EMPTY, EMPTY)
764
765 /* 0xa1 */
766    EMPTY, EMPTY, EMPTY, EMPTY)
767
768 /* 0xa2 */
769    EMPTY, EMPTY, EMPTY, EMPTY)
770
771 /* 0xa3 */
772    EMPTY, EMPTY, EMPTY, EMPTY)
773
774 /* 0xa4 */
775    EMPTY, EMPTY, EMPTY, EMPTY)
776
777 /* 0xa5 */
778    EMPTY, EMPTY, EMPTY, EMPTY)
779
780 /* 0xa6 */
781    EMPTY, EMPTY, EMPTY, EMPTY)
782
783 /* 0xa7 */
784    EMPTY, EMPTY, EMPTY, EMPTY)
785
786 /* 0xa8 */
787    EMPTY, EMPTY, EMPTY, EMPTY)
788
789 /* 0xa9 */
790    EMPTY, EMPTY, EMPTY, EMPTY)
791
792 /* 0xa0 */
793    EMPTY, EMPTY, EMPTY, EMPTY)
794
795 /* 0xa1 */
796    EMPTY, EMPTY, EMPTY, EMPTY)
797
798 /* 0xa2 */
799    EMPTY, EMPTY, EMPTY, EMPTY)
800
801 /* 0xa3 */
802    EMPTY, EMPTY, EMPTY, EMPTY)
803
804 /* 0xa4 */
805    EMPTY, EMPTY, EMPTY, EMPTY)
806
807 /* 0xa5 */
808    EMPTY, EMPTY, EMPTY, EMPTY)
809
810 /* 0xa6 */
811    EMPTY, EMPTY, EMPTY, EMPTY)
812
813 /* 0xa7 */
814    EMPTY, EMPTY, EMPTY, EMPTY)
815
816 /* 0xa8 */
817    EMPTY, EMPTY, EMPTY, EMPTY)
818
819 /* 0xa9 */
820    EMPTY, EMPTY, EMPTY, EMPTY)
821
822 /* 0xa0 */
823    EMPTY, EMPTY, EMPTY, EMPTY)
824
825 /* 0xa1 */
826    EMPTY, EMPTY, EMPTY, EMPTY)
827
828 /* 0xa2 */
829    EMPTY, EMPTY, EMPTY, EMPTY)
830
831 /* 0xa3 */
832    EMPTY, EMPTY, EMPTY, EMPTY)
833
834 /* 0xa4 */
835    EMPTY, EMPTY, EMPTY, EMPTY)
836
837 /* 0xa5 */
838    EMPTY, EMPTY, EMPTY, EMPTY)
839
840 /* 0xa6 */
841    EMPTY, EMPTY, EMPTY, EMPTY)
842
843 /* 0xa7 */
844    EMPTY, EMPTY, EMPTY, EMPTY)
845
846 /* 0xa8 */
847    EMPTY, EMPTY, EMPTY, EMPTY)
848
849 /* 0xa9 */
850    EMPTY, EMPTY, EMPTY, EMPTY)
851
852 /* 0xa0 */
853    EMPTY, EMPTY, EMPTY, EMPTY)
854
855 /* 0xa1 */
856    EMPTY, EMPTY, EMPTY, EMPTY)
857
858 /* 0xa2 */
859    EMPTY, EMPTY, EMPTY, EMPTY)
860
861 /* 0xa3 */
862    EMPTY, EMPTY, EMPTY, EMPTY)
863
864 /* 0xa4 */
865    EMPTY, EMPTY, EMPTY, EMPTY)
866
867 /* 0xa5 */
868    EMPTY, EMPTY, EMPTY, EMPTY)
869
870 /* 0xa6 */
871    EMPTY, EMPTY, EMPTY, EMPTY)
872
873 /* 0xa7 */
874    EMPTY, EMPTY, EMPTY, EMPTY)
875
876 /* 0xa8 */
877    EMPTY, EMPTY, EMPTY, EMPTY)
878
879 /* 0xa9 */
880    EMPTY, EMPTY, EMPTY, EMPTY)
881
882 /* 0xa0 */
883    EMPTY, EMPTY, EMPTY, EMPTY)
884
885 /* 0xa1 */
886    EMPTY, EMPTY, EMPTY, EMPTY)
887
888 /* 0xa2 */
889    EMPTY, EMPTY, EMPTY, EMPTY)
890
891 /* 0xa3 */
892    EMPTY, EMPTY, EMPTY, EMPTY)
893
894 /* 0xa4 */
895    EMPTY, EMPTY, EMPTY, EMPTY)
896
897 /* 0xa5 */
898    EMPTY, EMPTY, EMPTY, EMPTY)
899
900 /* 0xa6 */
901    EMPTY, EMPTY, EMPTY, EMPTY)
902
903 /* 0xa7 */
904    EMPTY, EMPTY, EMPTY, EMPTY)
905
906 /* 0xa8 */
907    EMPTY, EMPTY, EMPTY, EMPTY)
908
909 /* 0xa9 */
910    EMPTY, EMPTY, EMPTY, EMPTY)
911
912 /* 0xa0 */
913    EMPTY, EMPTY, EMPTY, EMPTY)
914
915 /* 0xa1 */
916    EMPTY, EMPTY, EMPTY, EMPTY)
917
918 /* 0xa2 */
919    EMPTY, EMPTY, EMPTY, EMPTY)
920
921 /* 0xa3 */
922    EMPTY, EMPTY, EMPTY, EMPTY)
923
924 /* 0xa4 */
925    EMPTY, EMPTY, EMPTY, EMPTY)
926
927 /* 0xa5 */
928    EMPTY, EMPTY, EMPTY, EMPTY)
929
930 /* 0xa6 */
931    EMPTY, EMPTY, EMPTY, EMPTY)
932
933 /* 0xa7 */
934    EMPTY, EMPTY, EMPTY, EMPTY)
935
936 /* 0xa8 */
937    EMPTY, EMPTY, EMPTY, EMPTY)
938
939 /* 0xa9 */
940    EMPTY, EMPTY, EMPTY, EMPTY)
941
942 /* 0xa0 */
943    EMPTY, EMPTY, EMPTY, EMPTY)
944
945 /* 0xa1 */
946    EMPTY, EMPTY, EMPTY, EMPTY)
947
948 /* 0xa2 */
949    EMPTY, EMPTY, EMPTY, EMPTY)
950
951 /* 0xa3 */
952    EMPTY, EMPTY, EMPTY, EMPTY)
953
954 /* 0xa4 */
955    EMPTY, EMPTY, EMPTY, EMPTY)
956
957 /* 0xa5 */
958    EMPTY, EMPTY, EMPTY, EMPTY)
959
960 /* 0xa6 */
961    EMPTY, EMPTY, EMPTY, EMPTY)
962
963 /* 0xa7 */
964    EMPTY, EMPTY, EMPTY, EMPTY)
965
966 /* 0xa8 */
967    EMPTY, EMPTY, EMPTY, EMPTY)
968
969 /* 0xa9 */
970    EMPTY, EMPTY, EMPTY, EMPTY)
971
972 /* 0xa0 */
973    EMPTY, EMPTY, EMPTY, EMPTY)
974
975 /* 0xa1 */
976    EMPTY, EMPTY, EMPTY, EMPTY)
977
978 /* 0xa2 */
979    EMPTY, EMPTY, EMPTY, EMPTY)
980
981 /* 0xa3 */
982    EMPTY, EMPTY, EMPTY, EMPTY)
983
984 /* 0xa4 */
985    EMPTY, EMPTY, EMPTY, EMPTY)
986
987 /* 0xa5 */
988    EMPTY, EMPTY, EMPTY, EMPTY)
989
990 /* 0xa6 */
991    EMPTY, EMPTY, EMPTY, EMPTY)
992
993 /* 0xa7 */
994    EMPTY, EMPTY, EMPTY, EMPTY)
995
996 /* 0xa8 */
997    EMPTY, EMPTY, EMPTY, EMPTY)
998
999 /* 0xa9 */
1000   EMPTY, EMPTY, EMPTY, EMPTY)

```

```

(nemu) c
[src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 17:19:51, May 28 2020
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x100cf0, end = 0x1052cc,
size = 17884 bytes
invalid opcode(eip = 0x04001f98): cd 80 5b 5d c3 66 90 90 ...
There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x04001f98 is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x04001f98) in the disassembling result to distinguish which case
it is.

```

2.2 触发异常

触发异常后硬件处理过程如下，

- 依次将 `EFLAGS`, `CS`, `EIP` 寄存器压入堆栈
- 从 `IDTR` 中读出 `IDT` 的首地址
- 根据异常（中断）号在 `IDT` 中进行索引，找到一个门的描述符
- 将门描述符中的 `offset` 域组合成目标地址
- 跳转到目标地址

代码实现过程如下。

其一，`CS` 寄存器的注册和初始化。

在 `nemu/include/cpu/reg.h` 中注册 `CS` 寄存器如下。

```
55      // 实现cs寄存器  
56      rtlreg_t cs;
```

在nemu/src/monitor/monitor.c中初始化cs寄存器如下。

```
83 static inline void restart() {  
84     /* Set the initial instruction pointer. */  
85     cpu.eip = ENTRY_START;  
86     // 进行eflags的初始化  
87     unsigned int origin = 2;  
88     memcpy(&cpu.eflags, &origin, sizeof(cpu.eflags));  
89     // 对cs寄存器初始化  
90     cpu.cs = 8;  
91  
92 #ifdef DIFF_TEST  
93     init_qemu_reg();  
94 #endif  
95 }
```

其二，实现int指令。

在nemu/src/cpu/intr.c中实现raise_intr函数，用于触发异常后的硬件处理如下。

```
4 void raise_intr(uint8_t NO, vaddr_t ret_addr) {  
5     /* TODO: Trigger an interrupt/exception with ``NO''.  
6     * That is, use ``NO'' to index the IDT.  
7     */  
8  
9     // TODO():  
10    //依次将eflags,cs,eip入栈  
11    memcpy(&t1, &cpu.eflags, sizeof(cpu.eflags));  
12    rtl_li(&t0, t1);  
13    rtl_push(&t0); //eflags  
14    rtl_push(&cpu.cs); //cs  
15    rtl_li(&t0, ret_addr);  
16    rtl_push(&t0); //eip  
17  
18    // 找到IDT中NO对应的门描述符首地址  
19    vaddr_t gate_addr = cpu.idtr.base + NO * sizeof(GateDesc);  
20    assert(gate_addr <= cpu.idtr.base + cpu.idtr.limit);  
21  
22    //读取门描述符的offset，计算目标地址  
23    uint32_t off_15_0 = vaddr_read(gate_addr, 2);  
24    uint32_t off_32_16 = vaddr_read(gate_addr + sizeof(GateDesc) - 2, 2);  
25    uint32_t target_addr = (off_32_16 << 16) + off_15_0;  
26 #ifdef DEBUG  
27     Log("target_addr=0x%08x", target_addr);  
28 #endif  
29    decoding.is_jmp = 1;  
30    decoding.jmp_eip = target_addr;  
31 }
```

在nemu/src/cpu/exec/system.c中实现执行函数如下。

```
41 extern void raise_intr(uint8_t NO, vaddr_t ret_addr);  
42 make_EHelper(int) {  
43     // TODO();  
44  
45     uint8_t NO = id_dest->val & 0xff;  
46     raise_intr(NO, decoding.seq_eip);  
47  
48     print_asm("int %s", id_dest->str);  
49  
50 #ifdef DIFF_TEST  
51     diff_test_skip_nemu();  
52 #endif  
53 }
```

在opcode_table中注册指令如下（位于1字节部分）。

```
126     /* 0xcc */     EMPTY, INDEXW(I,int,1), EMPTY, EMPTY,
```

最后，在`nanos-lite`目录下执行命令`make run`，运行dummy程序如下。

```
(nemu) c
[!]src/main.c,19,main] 'Hello World!' from Nanos-lite
[!]src/main.c,20,main] Build time: 19:51:32, May 31 2020
[!]src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x100ef4, end = 0x1054d0,
size = 17884 bytes
[!]src/main.c,27,main] Initializing interrupt/exception handler...
[!]src/cpu/exec/system.c,18,exec_lidt] idtr.limit=0x7ff
[!]src/cpu/exec/system.c,19,exec_lidt] idtr.base=0x105500
[!]src/cpu/intr.c,27,raise_intr] target_addr=0x100b17
invalid opcode(eip = 0x00100b26): 60 54 e8 03 ff ff ff 83 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x00100b26 is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x00100b26) in the disassembling result to distinguish which case it is.
```

2.3 保存现场

首先需要实现`pusha`指令，查阅i386手册，手册中关于该指令的说明如下。

PUSHA/PUSHAD — Push all General Registers

Opcode	Instruction	Clocks	Description
60	PUSHA	18	Push AX, CX, DX, BX, original SP, BP, SI, and DI
60	PUSHAD	18	Push EAX, ECX, EDX, EBX, original ESP, EBP, ESI, and EDI

Operation

```
IF OperandSize = 16 (* PUSHA instruction *)
THEN
    Temp ← (SP);
    Push(AX);
    Push(CX);
    Push(DX);
    Push(BX);
    Push(Temp);
    Push(BP);
    Push(SI);
    Push(DI);
ELSE (* OperandSize = 32, PUSHAD instruction *)
    Temp ← (ESP);
    Push(EAX);
    Push(ECX);
    Push(EDX);
    Push(EBX);
    Push(Temp);
    Push(EBP);
    Push(ESI);
    Push(EDI);
FI;
```

在`nemu/src/cpu/exec/data-mov.c`中添加执行函数如下。

```
23 make_EHelper(pusha) {
24     //TODO();
25
26     t0 = cpu.esp;
27     rtl_push(&cpu.eax);
28     rtl_push(&cpu.ecx);
29     rtl_push(&cpu.edx);
30     rtl_push(&cpu.ebx);
31     rtl_push(&t0);
32     rtl_push(&cpu.ebp);
33     rtl_push(&cpu.esi);
34     rtl_push(&cpu.edi);
35
36     print_asm("pusha");
37 }
```

补充opcode表如下。

```
166 /* 0x60 */ EX(pusha), EMPTY, EMPTY, EMPTY,
```

接下来实现_RegSet，由于nemu中栈从高地址向低地址生长，`struct _RegSet`结构中的内容从低地址向高地址延伸，所以先入栈的后声明，后入栈的先声明。

在 `nexus-am/am/arch/x86-nemu/include/arch.h` 中添加如下代码。

```
9 struct _RegSet {
10     //uintptr_t esi, ebx, eax, eip, edx, error_code, eflags, ecx, cs, esp, e
11     di, ebp;
12     //int      irq;
13     uintptr_t edi, esi, ebp, esp, ebx, edx, ecx, eax;
14     int irq;
15     uintptr_t error_code;
16     uintptr_t eip;
17     uintptr_t cs;
18     uintptr_t eflags;
19};
```

在 `nanos-lite` 下执行 `make run` 如下。

```
(nemu) c
[0]src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 19:51:32, May 31 2020
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x100ef4, end = 0x1054d0,
size = 17884 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
[src/cpu/exec/system.c,18,exec_lidt] idtr.limit=0x7ff
[src/cpu/exec/system.c,19,exec_lidt] idtr.base=0x105500
[src/cpu/intr.c,27,raise_intr] target_addr=0x100b17
[src/irq.c,5,do_event] system panic: Unhandled event ID = 8
nemu: HIT BAD TRAP at eip = 0x00100032
```

思考题一：对比异常与函数调用

对比异常与函数调用

我们知道进行函数调用的时候也需要保存调用者的状态：返回地址，以及调用约定(calling convention)中需要调用者保存的寄存器。而进行异常处理之前却要保存更多的信息。尝试对比它们，并思考两者保存信息不同是什么原因造成的。

根据实验指导，call分伪近调用和远调用，远调用压段寄存器cs和返回地址eip；近调用只压入返回地址eip。而异常处理要压入eflags, cs, eip，通用寄存器的所有信息。对于call操作保存现场方式，所有需要预存的参数，都需要汇编语言程序员手动进行入栈和出栈，即保存现场和恢复现场，即程序员主动保存参数，主动调用子程序，主动恢复现场。

call和异常保存信息的不同主要源于服务时间与服务对象的不同。一方面，调用子程序过程发生的时间是已知和固定的，在主程序中的调用指令(call)执行时发生，而中断/异常发生的时间一般是随机的，调用子程序是程序设计者事先安排的，而执行中断服务程序是由系统工作环境随机决定的。另一方面，子程序完全为主程序服务，二者为主从关系，而中断服务程序与主程序之间一般是无关的，二者是独立平行的关系。

思考题二：诡异的代码

诡异的代码

trap.S中有一行 `pushl %esp` 的代码，乍看之下其行为十分诡异。你能结合前后的代码理解它的行为吗？Hint：不用想太多，其实都是你学过的知识。

`pushl %esp` 代码是在执行压入参数的过程，其目的是将TrapFrame的首地址作为参数传递给irq_handle函数的。从irq_handle函数获取到参数 `_RegSet *tf` 参数的过程即可看出，代码如下。

```

_RegSet* irq_handle(_RegSet *tf) {
    _RegSet *next = tf;
    if (H) {
        _Event ev;
        switch (tf->irq) {
            case 0x80: ev.event = _EVENT_SYSCALL; break;
            default: ev.event = _EVENT_ERROR; break;
        }
    }

    next = H(ev, tf);
    if (next == NULL) {
        next = tf;
    }
}

return next;
}

```

2.4 事件分发

在 `nanos-lite/src/irq.c` 中添加代码，用来识别系统调用事件如下。

```

3 extern _RegSet* do_syscall(_RegSet *r);
4 static _RegSet* do_event(_Event e, _RegSet* r) {
5     switch (e.event) {
6         case _EVENT_SYSCALL:
7             return do_syscall(r);
8         default: panic("Unhandled event ID = %d", e.event);
9     }
10
11     return NULL;
12 }

```

运行代码，结果如下。

```

(nemu) c
[0]src/main.c,19,main] 'Hello World!' from Nanos-lite
[0]src/main.c,20,main] Build time: 19:51:32, May 31 2020
[0]src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x100f98, end = 0x105574,
size = 17884 bytes
[0]src/main.c,27,main] Initializing interrupt/exception handler...
[0]src/cpu/exec/system.c,18,exec_lidt] idtr.limit=0x7ff
[0]src/cpu/exec/system.c,19,exec_lidt] idtr.base=0x1055a0
[0]src/cpu/intc.c,27,raise_intr] target_addr=0x100b5b
[0]src/syscall.c,9,do_syscall] system panic: Unhandled syscall ID = 0
nemu: HIT BAD TRAP at eip = 0x00100032

```

可以看到，这里指示的 `syscall ID=0`，即 dummy 触发的号码为 0 的 `SYS_none` 系统调用。

2.5 系统调用处理

根据实验指导，对于宏 `SYSCALL_ARGx`，其任务是从 TrapFrame 的寄存器中读取参数，因此在 `nexus-am/am/arch/x86-nemu/include/arch.h` 中修改代码如下。

```

21 #define SYSCALL_ARG1(r) r->eax
22 #define SYSCALL_ARG2(r) r->ebx
23 #define SYSCALL_ARG3(r) r->ecx
24 #define SYSCALL_ARG4(r) r->edx

```

接下来需要添加系统调用 `SYS_none` 与处理函数 `sys_none`。

在 `nanos-lite/src/syscall.c` 中添加代码如下。

```
4 int sys_none(){
5     return 1;
6 }
7
8 void sys_exit(int a){
9     _halt(a);
10}
11
12 _RegSet* do_syscall(_RegSet *r) {
13     uintptr_t a[4];
14     a[0] = SYSCALL_ARG1(r);
15     a[1] = SYSCALL_ARG2(r);
16     a[2] = SYSCALL_ARG3(r);
17     a[3] = SYSCALL_ARG4(r);
18
19     switch (a[0]) {
20         case SYS_none:SYSCALL_ARG1(r) = sys_none();break;
21         case SYS_exit:sys_exit(a[1]);break;
22         default: panic("Unhandled syscall ID = %d", a[0]);
23     }
24
25     return NULL;
26 }
```

运行代码，结果如下。

```
(nemu) c
[0]src/main.c,19,main] 'Hello World!' from Nanos-lite
[0]src/main.c,20,main] Build time: 20:37:22, May 31 2020
[0]src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x100fe4, end = 0x1055c0,
size = 17884 bytes
[0]src/main.c,27,main] Initializing interrupt/exception handler...
[0]src/cpu/exec/system.c,18,exec_lidt] idtr.limit=0x7ff
[0]src/cpu/exec/system.c,19,exec_lidt] idtr.base=0x1055e0
[0]src/cpu/intr.c,27,raise_intr] target_addr=0x100ba7
invalid opcode(eip = 0x00100bc0): 61 83 c4 08 cf 00 00 00 ...
There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x00100bc0 is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x00100bc0) in the disassembling result to distinguish which case it is.
```

注意到 `popa` 指令尚未实现。

2.6 恢复现场

查阅 i386 手册中关于 `popa` 的内容如下。

POPA/POPAD — Pop all General Registers

Opcode	Instruction	Clocks	Description
61	POPA	24	Pop DI, SI, BP, SP, BX, DX, CX, and AX
61	POPAD	24	Pop EDI, ESI, EBP, ESP, EDX, ECX, and EAX

Operation

```
IF OperandSize = 16 (* instruction = POPA *)
THEN
    DI ← Pop();
    SI ← Pop();
    BP ← Pop();
    throwaway ← Pop (); (* Skip SP *)
    BX ← Pop();
    DX ← Pop();
    CX ← Pop();
    AX ← Pop();
ELSE (* OperandSize = 32, instruction = POPAD *)
    EDI ← Pop();
    ESI ← Pop();
    EBP ← Pop();
    throwaway ← Pop (); (* Skip ESP *)
    EBX ← Pop();
    EDX ← Pop();
    ECX ← Pop();
    EAX ← Pop();
FI;
```

在 `nemu/src/cpu/exec/data-mov.c` 中添加 pop 指令的执行函数如下。

```
39 make_EHelper(popa) {
40     //TODO();
41
42     rtl_pop(&cpu.edi);
43     rtl_pop(&cpu.esi);
44     rtl_pop(&cpu.ebp);
45     rtl_pop(&t0);
46     rtl_pop(&cpu.ebx);
47     rtl_pop(&cpu.edx);
48     rtl_pop(&cpu.ecx);
49     rtl_pop(&cpu.eax);
50
51     print_asm("popa");
52 }
```

注册 opcode 信息如下。

```
99 /* 0x60 */ EX(pusha), EX(popa), EMPTY, EMPTY,
```

接下来需要继续实现 iret 指令。

查阅 i386 手册中的内容如下。

IRET/IREDT — Interrupt Return

Opcode	Instruction	Clocks	Description
CF	IRET	22,pm=38	Interrupt return (far return and pop flags)
CF	IRET	pm=82	Interrupt return to lesser privilege
CF	IRET	ts	Interrupt return, different task (NT = 1)
CF	IREDT	22,pm=38	Interrupt return (far return and pop flags)
CF	IREDT	pm=82	Interrupt return to lesser privilege
CF	IREDT	pm=60	Interrupt return to V86 mode
CF	IREDT	ts	Interrupt return, different task (NT = 1)

NOTE:

Values of ts are given by the following table:

New Task			
Old Task	386 TSS VM = 0	386 TSS VM = 1	286 TSS
386 TSS VM=0	275	224	271
286 TSS	265	214	232

在 `nemu/src/cpu/exec/exec.c` 中添加执行函数如下。

```
55 make_EHelper(iret) {
56     //TODO();
57
58     rtl_pop(&cpu.eip);
59     rtl_pop(&cpu.cs);
60     rtl_pop(&t0);
61     memcpy(&cpu.eflags, &t0, sizeof(cpu.eflags));
62     decoding.jmp_eip = 1;
63     decoding.seq_eip = cpu.eip;
64
65     print_asm("iret");
66 }
```

补充opcode表如下。

```
126 /* 0xcc */    EMPTY, INDEXW(I,int,1), EMPTY, EX(iret),
```

运行程序，结果如下。

```
(nemu) c
[0]src/main.c,19,main] 'Hello World!' from Nanos-lite
[0]src/main.c,20,main] Build time: 20:37:22, May 31 2020
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x100fe4, end = 0x1055c0,
size = 17884 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
[src/cpu/exec/system.c,18,exec_lidt] idtr.limit=0x7ff
[src/cpu/exec/system.c,19,exec_lidt] idtr.base=0x1055e0
[src/cpu/intr.c,27,raise_intr] target_addr=0x100ba7
[src/cpu/intr.c,27,raise_intr] target_addr=0x100ba7
nemu: HIT GOOD TRAP at eip = 0x00100032
```

以上是本次实验阶段一部分。

PA3：阶段二

阶段二主要内容是进一步完善系统调用，实现简易文件系统。

3.1 代码：在Nanos-lite上运行Hello world

在 Nanos-lite 上运行 Hello world

实现 write() 系统调用，然后把 Nanos-lite 上运行的用户程序切换成 hello 程序并运行：

- ❖ 切换到 navy-apps/tests/hello/ 目录下执行 make 编译 hello 程序
- ❖ 修改 nanos-lite/Makefile 中 ramdisk 的生成规则，把 ramdisk 中的唯一的文件换成 hello 程序：

```
-- nanos-lite/Makefile
+++ nanos-lite/Makefile
@@ -9,2 +9,2 @@
OBJCOPY_FLAG = -S --set-section-flags .bss=alloc,contents -O binary
-OBJCOPY_FILE = $(NAVY_HOME)/tests/dummy/build/dummy-x86
+OBJCOPY_FILE = $(NAVY_HOME)/tests/hello/build/hello-x86
```
- ❖ 在 nanos-lite/Makefile 下执行 make update 更新 ramdisk
- ❖ 重新编译 Nanos-lite 并运行

首先实现 SYS_write 的系统调用部分。

在 `nanos-lite/src/syscall.c` 中添加如下代码。这里需要注意，对于 `stdout` 与 `stderr` 暂时不考虑物理空间不足等造成实际写字节数小于请求字节数的情况。

```
12 int sys_write(int fd, void* buf, size_t len){
13     if(fd == 1 || fd == 2){
14         char c;
15         for(int i = 0; i < len; i++){
16             memcpy(&c, buf + i, 1);
17             _putc(c);
18         }
19         return len;
20     }
21     else
22         panic("Unhandled fd = %d in sys_write", fd);
23     return -1;
24 }
```



```
33     switch (a[0]) {
34         case SYS_none:SYSCALL_ARG1(r) = sys_none();break;
35         case SYS_exit:sys_exit(a[1]);break;
36         case SYS_write:SYSCALL_ARG1(r) = sys.write(a[1], (void*)a[2], a[3]);break;
37         default: panic("Unhandled syscall ID = %d", a[0]);
38     }
```

在 `navy-apps/libs/libos/src/nanos.c` 中添加辅助函数如下。

```
28 int _write(int fd, void *buf, size_t count){
29     /*_exit(SYS_write);
30     return _syscall_(SYS_write, fd, (uintptr_t)buf, count);
31 }
```

运行代码，结果如下。

```
(nemu) c
[0]src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 20:37:22, May 31 2020
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x1010e8, end = 0x1057c4, size = 1814
0 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
[src/cpu/exec/system.c,18,exec_lidt] idtr.limit=0x7ff
[src/cpu/exec/system.c,19,exec_lidt] idtr.base=0x105800
Hello World!
Hello World for the 2th time
```

3.2 堆区管理

首先，实现 SYS_brk 系统调用。该调用会接受一个参数 `addr`，用于指示新的 program break 的位置，目前 Nanos-lite 为单任务 OS，用户程序可自由使用空闲的内存，所以只要让 `SYS_brk` 总是返回 0，表示堆区大小的调整总是成功。

于是，在 `nanos-lite/src/syscall.c` 中添加如下代码。

```

26 int sys_brk(int addr){
27     return 0;
28 }
29
30 _RegSet* do_syscall(_RegSet *r) {
31     uintptr_t a[4];
32     a[0] = SYSCALL_ARG1(r);
33     a[1] = SYSCALL_ARG2(r);
34     a[2] = SYSCALL_ARG3(r);
35     a[3] = SYSCALL_ARG4(r);
36
37     switch (a[0]) {
38         case SYS_none:SYSCALL_ARG1(r) = sys_none();break;
39         case SYS_exit:sys_exit(a[1]);break;
40         case SYS_write:sys_write(a[1], (void*)a[2], a[3]);break;
41         case SYS_brk:SYSCALL_ARG1(r) = sys_brk(a[1]);break;
42         default: panic("Unhandled syscall ID = %d", a[0]);
43     }
}

```

接下来，实现_sbrk，_sbrk使用记录的方式来管理program break，在实际代码实现中，使用static来记录program break，静态局部变量可保证始终驻留在全局数据区，直到程序运行结束，以及在编译时赋初值（仅一次），以后每次调用函数时不再重新赋初值而是保留上次函数调用结束时的值。

在 `navy-apps/libs/libos/src/nanos.c` 中添加如下代码。

```

33 void *_sbrk(intptr_t increment){
34     extern end;
35     static uintptr_t probreak = (uintptr_t)&end;
36     uintptr_t probreak_new = probreak + increment;
37     int r = _syscall_(SYS_brk, probreak_new, 0, 0);
38     if(r == 0){
39         uintptr_t temp = probreak;
40         probreak = probreak_new;
41         return (void*)temp;
42     }
43
44     return (void *)-1;
45 }

```

根据老师给的实验指导，为了比对运行前后效果，在`sys_write`中加入`Log()`观察`write`系统调用的调用情况如下。

```

12 int sys_write(int fd, void* buf, size_t len){
13     if(fd == 1 || fd == 2){
14         char c;
15         Log("buffer:%s", (char*)buf);
16         for(int i = 0; i < len; i++){

```

运行程序，结果如下。

代码实现前：

```

(nemu) c
[0]src/main.c,19,main] 'Hello World!' from Nanos-lite
[0]src/main.c,20,main] Build time: 20:37:22, May 31 2020
[0]src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x101154, end = 0x105830, size = 1814
0 bytes
[0]src/main.c,27,main] Initializing interrupt/exception handler...
[0]src/cpu/exec/system.c,18,exec_lidt] idtr.limit=0x7ff
[0]src/cpu/exec/system.c,19,exec_lidt] idtr.base=0x105860
[0]src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[0]src/syscall.c,15,sys_write] buffer:Hello World!

Hello World!
[0]src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[0]src/syscall.c,15,sys_write] buffer:H
H[0]src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[0]src/syscall.c,15,sys_write] buffer:e
e[0]src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[0]src/syscall.c,15,sys_write] buffer:l
l[0]src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[0]src/syscall.c,15,sys_write] buffer:l
l[0]src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[0]src/syscall.c,15,sys_write] buffer:o
o[0]src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[0]src/syscall.c,15,sys_write] buffer:
[0]src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b

```

代码实现后：

```
(nemu) c
[0]src/main.c,19,main] 'Hello World!' from Nanos-lite
[src/main.c,20,main] Build time: 20:37:22, May 31 2020
[src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x101154, end = 0x105874, size = 1820
8 bytes
[src/main.c,27,main] Initializing interrupt/exception handler...
[src/cpu/exec/system.c,18,exec_lidt] idtr.limit=0x7ff
[src/cpu/exec/system.c,19,exec_lidt] idtr.base=0x1058a0
[src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[src/syscall.c,15,sys_write] buffer:Hello World!

Hello World!
[src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[src/syscall.c,15,sys_write] buffer:Hello World for the 2th time

Hello World for the 2th time
[src/cpu/intr.c,27,raise_intr] target_addr=0x100c9b
[src/syscall.c,15,sys_write] buffer:Hello World for the 3th time

Hello World for the 3th time
```

3.3 简易文件系统

首先，需要先定义一些辅助函数，用于读取或设置Finfo结构中的变量。

在`nanos-lite/src/fs.c`中添加代码如下。

```
28 size_t fs_filesz(int fd){
29     assert(fd >= 0 && fd < NR_FILES);
30     return file_table[fd].size;
31 }
32
33 off_t disk_offset(int fd){
34     assert(fd >= 0 && fd < NR_FILES);
35     return file_table[fd].disk_offset;
36 }
37
38 off_t get_open_offset(int fd){
39     assert(fd >= 0 && fd < NR_FILES);
40     return file_table[fd].open_offset;
41 }
42
43 void set_open_offset(int fd, off_t n){
44     assert(fd >= 0 && fd < NR_FILES);
45     assert(n >= 0);
46     if(n > file_table[fd].size)
47         n = file_table[fd].size;
48     file_table[fd].open_offset = n;
49 }
```

`extern` 定义 ramdisk 的辅助函数如下。

```
51 extern void ramdisk_read(void *buf, off_t offset, size_t len);
52 extern void ramdisk_write(void *buf, off_t offset, size_t len);
```

实现`fs_open()`、`fs_read()`、`fs_close()`函数如下。

```

54 int fs_open(const char* filename, int flags, int mode){
55     for(int i = 0; i < NR_FILES; i++){
56         if(strcmp(filename, file_table[i].name) == 0)
57             return i;
58     }
59     panic("this filename not exist in file_table");
60     return -1;
61 }
62
63 ssize_t fs_read(int fd, void* buf, size_t len){
64     assert(fd >= 0 && fd < NR_FILES);
65     if(fd < 3){
66         Log("arg invalid:fd<3");
67         return 0;
68     }
69
70     int n = fs_filesz(fd) - get_open_offset(fd);
71     if(n > len)
72         n = len;
73     ramdisk_read(buf, disk_offset(fd) + get_open_offset(fd), n);
74     set_open_offset(fd, get_open_offset(fd) + n);
75     return n;
76 }
77
78 int fs_close(int fd){
79     assert(fd >= 0 && fd < NR_FILES);
80     return 0;
81 }

```

在文件 `nanos-lite/include/fs.h` 中添加如下代码。

```

1 size_t fs_filesz(int fd);
2
3 int fs_open(const char* filename, int flags, int mode);
4 ssize_t fs_read(int fd, void* buf, size_t len);
5 ssize_t fs_write(int fd, void* buf, size_t len);
6 int fs_close(int fd);
7 off_t fs_lseek(int fd, off_t offset, int whence);

```

在 `nanos-lite/src/loader.c` 中添加头文件并补充函数如下。

```

1 #include "common.h"
2 #include "fs.h"
3
4 #define DEFAULT_ENTRY ((void *)0x40000000)
5
6 // PA3-1
7 extern uint8_t ramdisk_start;
8 extern uint8_t ramdisk_end;
9 #define RAMDISK_SIZE ((&ramdisk_end) - (&ramdisk_start))
10 extern void ramdisk_read(void *buf, off_t offset, size_t len);
11
12 uintptr_t loader(_Protect *as, const char *filename) {
13     // TODO();
14     //ramdisk_read(DEFAULT_ENTRY, 0, RAMDISK_SIZE);
15
16     int fd = fs_open(filename, 0, 0);
17     Log("filename=%s, fd=%d", filename, fd);
18     fs_read(fd, DEFAULT_ENTRY, fs_filesz(fd));
19     fs_close(fd);
20     return (uintptr_t)DEFAULT_ENTRY;
21 }

```

在 `nanos-lite/src/main.c` 中修改代码如下。

```

33     uint32_t entry = loader(NULL, "/bin/text");
34     ((void (*)(void))entry)();

```

运行结果如下。

```
(nemu) c
[0]src/main.c,19,main] 'Hello World!' from Nanos-lite
[0]src/main.c,20,main] Build time: 22:50:30, May 31 2020
[0]src/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x1018e0, end = 0x3702ea, size = 2550
282 bytes
[0]src/main.c,27,main] Initializing interrupt/exception handler...
[0]src/cpu/exec/system.c,18,exec_lidt] idtr.limit=0x7ff
[0]src/cpu/exec/system.c,19,exec_lidt] idtr.base=0x370540
[0]src/loader.c,17,loader] filename=/bin/text, fd=24
[0]src/cpu/intr.c,27,raise_intr] target_addr=0x101007
[0]src/cpu/intr.c,27,raise_intr] target_addr=0x101007
[0]src/cpu/intr.c,27,raise_intr] target_addr=0x101007
[0]src/cpu/intr.c,27,raise_intr] target_addr=0x101007
[0]nemu: HIT BAD TRAP at eip = 0x00100032
```

3.4 实现完整文件系统

首先在 `nanos-lite/src/fs.c` 下添加 `fs_write` 函数如下。

```
79 ssize_t fs_write(int fd, void* buf, size_t len){
80     assert(fd >= 0 && fd < NR_FILES);
81     if(fd < 3){
82         Log("arg invaid:fd<3");
83         return 0;
84     }
85     int n = fs_filesz(fd) - get_open_offset(fd);
86     if(n > len){
87         n = len;
88     }
89     ramdisk_write(buf, disk_offset(fd) + get_open_offset(fd), n);
90     set_open_offset(fd, get_open_offset(fd) + n);
91     return n;
92 }
```

实现 `fs_lseek` 函数如下。

```
99 off_t fs_lseek(int fd, off_t offset, int whence){
100     switch(whence){
101         case SEEK_SET:
102             set_open_offset(fd, offset);
103             return get_open_offset(fd);
104         case SEEK_CUR:
105             set_open_offset(fd, get_open_offset(fd) + offset);
106             return get_open_offset(fd);
107         case SEEK_END:
108             set_open_offset(fd, fs_filesz(fd) + offset);
109             return get_open_offset(fd);
110         default:
111             panic("Unhandled whence ID = %d", whence);
112             return -1;
113     }
114 }
```

在 `nanos-lite/src/syscall1.c` 中修改系统调用如下。

```
12 int sys_write(int fd, void* buf, size_t len){
13     if(fd == 1 || fd == 2){
14         char c;
15         //Log("buffer:%s", (char*)buf);
16         for(int i = 0; i < len; i++){
17             memcpy(&c, buf + i, 1);
18             _putc(c);
19         }
20         return len;
21     }
22     else
23         panic("Unhandled fd = %d in sys_write", fd);
24
25     if(fd >= 3)
26         return fs_write(fd, buf, len);
27     Log("fd<0");
28     return -1;
29 }
```

```

35 int sys_open(const char* filename){
36     return fs_open(filename, 0, 0);
37 }
38
39 int sys_read(int fd, void* buf, size_t len){
40     return fs_read(fd, buf, len);
41 }
42
43 int sys_close(int fd){
44     return fs_close(fd);
45 }
46
47 int sys_lseek(int fd, off_t offset, int whence){
48     return fs_lseek(fd, offset, whence);
49 }

58     switch (a[0]) {
59         case SYS_none:SYSCALL_ARG1(r) = sys_none();break;
60         case SYS_exit:sys_exit(a[1]);break;
61         case SYS_write:SYSCALL_ARG1(r) = sys_write(a[1], (void*)a[2], a[3]);break;
62         case SYS_brk:SYSCALL_ARG1(r) = sys_brk(a[1]);break;
63         case SYS_open:SYSCALL_ARG1(r) = sys_open((char*)a[1], (void*)a[2], a[3]);break;
64         case SYS_read:SYSCALL_ARG1(r) = sys_read(a[1], (void*)a[2], a[3]);break;
65         case SYS_close:SYSCALL_ARG1(r) = sys_close(a[1]);break;
66         case SYS_lseek:SYSCALL_ARG1(r) = sys_lseek(a[1], a[2], a[3]);break;
67         default: panic("Unhandled syscall ID = %d", a[0]);
68     }

```

在 `navy-apps/libs/libos/src/nanos.c` 中添加如下函数。

```

24 int _open(const char *path, int flags, mode_t mode) {
25     //_exit(SYS_open);
26     return _syscall_(SYS_open, (uintptr_t)path, flags, mode);
27 }

48 int _read(int fd, void *buf, size_t count) {
49     //_exit(SYS_read);
50     return _syscall_(SYS_read, fd, (uintptr_t)buf, count);
51 }
52
53 int _close(int fd) {
54     //_exit(SYS_close);
55     return _syscall_(SYS_close, fd, 0, 0);
56 }
57
58 off_t _lseek(int fd, off_t offset, int whence) {
59     //_exit(SYS_lseek);
60     return _syscall_(SYS_lseek, fd, offset, whence);
61 }

```

最后，运行代码，发现依然出现 Hit Bad Trap 问题，经过 debug，修改 `sys_write()` 函数如下。

```

13 int sys_write(int fd, void* buf, size_t len){
14     if(fd == 1 || fd == 2){
15         char c;
16         //Log("buffer:%s", (char*)buf);
17         for(int i = 0; i < len; i++){
18             memcpy(&c, buf + i, 1);
19             _putc(c);
20         }
21         return len;
22     }
23
24     if(fd >= 3)
25         return fs_write(fd, buf, len);
26     Log("fd<=0");
27     return -1;
28 }

```

再次运行代码，花了很长时间，编译通过，输出 PASS! ! !

以上是本次实验阶段二部分内容。

PA3: 阶段三

阶段三的主要内容是将输入输出抽象成文件，并运行仙剑奇侠传。

4.1 代码：把VGA显存抽象成文件

把VGA显存抽象成文件

你需要在 Nanos-lite 中

- 在 `init_fs()` (在 `nanos-lite/src/fs.c` 中定义) 中对文件记录表中 `/dev/fb` 的大小进行初始化，你需要使用 IOE 定义的 API 来获取屏幕的大小。
- 实现 `fb_write()` (在 `nanos-lite/src/device.c` 中定义)，用于把 `buf` 中的 `len` 字节写到屏幕上 `offset` 处。你需要先从 `offset` 计算出屏幕上的坐标，然后调用 IOE 的 `_draw_rect()` 接口。
- 在 `init_device()` (在 `nanos-lite/src/device.c` 中定义) 中将 `/proc/dispinfo` 的内容提前写入到字符串 `dispinfo` 中。实际的屏幕大小信息已经记录在 AM 的 IOE 接口中，你需要在 Nanos-lite 中获取它们。
- 实现 `dispinfo_read()` (在 `nanos-lite/src/device.c` 中定义)，用于把字符串 `dispinfo` 中 `offset` 开始的 `len` 字节写到 `buf` 中。
- 在文件系统中添加对 `/dev/fb` 和 `/proc/dispinfo` 这两个特殊文件的支持。

让 Nanos-lite 加载 `/bin/bmp-test`，如果实现正确，你将会看到屏幕上显示 ProjectN 的 Logo。

首先，为了使得 nanos-lite 获得 AM 屏幕信息，需要在 IOE 中添加接口。在 `nexus-am/am/arch/x86-nemu/src/ioe.c` 中添加代码如下。

```
51 // 辅助函数，返回屏幕大小信息
52 void getScreen(int* width, int* height){
53     *width = _screen.width;
54     *height = _screen.height;
55 }
```

接下来在 `nanos-lite/src/fs.c` 中实现初始化函数如下。

```
25 void init_fs() {
26     // TODO: initialize the size of /dev/fb
27     extern void getScreen(int* p_width, int* p_height);
28     int width = 0, height = 0;
29     getScreen(&width, &height);
30     file_table[FD_FB].size = width * height * sizeof(uint32_t);
31     Log("set FD_FB size=%d", file_table[FD_FB].size);
32 }
```

在 `nanos-lite/src/device.c` 中添加如下代码。

```
21 void fb_write(const void *buf, off_t offset, size_t len) {
22     assert(offset % 4 == 0 && len % 4 == 0);
23     int index, screen_x1, screen_y1, screen_y2;
24     int width = 0, height = 0;
25     getScreen(&width, &height);
26
27     index = offset / 4;
28     screen_y1 = index / width;
29     screen_x1 = index % width;
30
31     index = (offset + len) / 4;
32     screen_y2 = index / width;
33
34     assert(screen_y2 >= screen_y1);
35
36     if(screen_y2 == screen_y1){
37         _draw_rect(buf, screen_x1, screen_y1, len / 4, 1);
38         return;
39     }
40 }
```

```

41     int tempw = width - screen_x1;
42     if(screen_y2 - screen_y1 == 1){
43         _draw_rect(buf, screen_x1, screen_y1, tempw, 1);
44         _draw_rect(buf + tempw * 4, 0, screen_y2, len / 4 - tempw, 1);
45         return;
46     }
47     _draw_rect(buf, screen_x1, screen_y1, tempw, 1);
48     int tempy = screen_y2 - screen_y1 - 1;
49     _draw_rect(buf + tempw * 4, 0, screen_y1 + 1, width, tempy);
50     _draw_rect(buf + tempw * 4 + tempy * width * 4, 0, screen_y2, len / 4 -
51     tempw - tempy * width, 1);
51 }

```

接下来，实现初始化设备的代码，在`nanos-lite/src/device.c`中如下。

```

54 void init_device() {
55     _ioe_init();
56
57     // TODO: print the string to array `dispinfo` with the format
58     // described in the Navy-apps convention
59     int width = 0, height = 0;
60     getScreen(&width, &height);
61     sprintf(dispinfo, "WIDTH:%d\nHEIGHT:%d\n", width, height);
62     // Log("%s", dispinfo);
63 }

17 void dispinfo_read(void *buf, off_t offset, size_t len) {
18     strncpy(buf, dispinfo + offset, len);
19 }

```

修改`fs.c`中代码如下。

```

69 void dispinfo_read(void *buf, off_t offset, size_t len);
70
71 ssize_t fs_read(int fd, void* buf, size_t len){
72     assert(fd >= 0 && fd < NR_FILES);
73     if(fd < 3 || fd == FD_FB){
74         Log("arg invalid:fd<3 || fd==FD_FB");
75         return 0;
76     }
77
78     int n = fs_filesz(fd) - get_open_offset(fd);
79     if(n > len)
80         n = len;
81     if(fd == FD_DISPINFO)
82         dispinfo_read(buf, get_open_offset(fd), n);
83     else
84         ramdisk_read(buf, disk_offset(fd) + get_open_offset(fd), n);
85     set_open_offset(fd, get_open_offset(fd) + n);
86     return n;
87 }

88 ssize_t fs_write(int fd, void* buf, size_t len){
89     assert(fd >= 0 && fd < NR_FILES);
90     if(fd < 3 || fd == DISPINFO){
91         Log("arg invalid:fd<3 || fd==DISPINFO");
92         return 0;
93     }
94     int n = fs_filesz(fd) - get_open_offset(fd);
95     if(n > len){
96         n = len;
97     }
98     if(fd == FD_FB)
99         fb_write(buf, get_open_offset(fd), n);
100    else
101        ramdisk_write(buf, disk_offset(fd) + get_open_offset(fd), n);
102    set_open_offset(fd, get_open_offset(fd) + n);
103    return n;
104 }

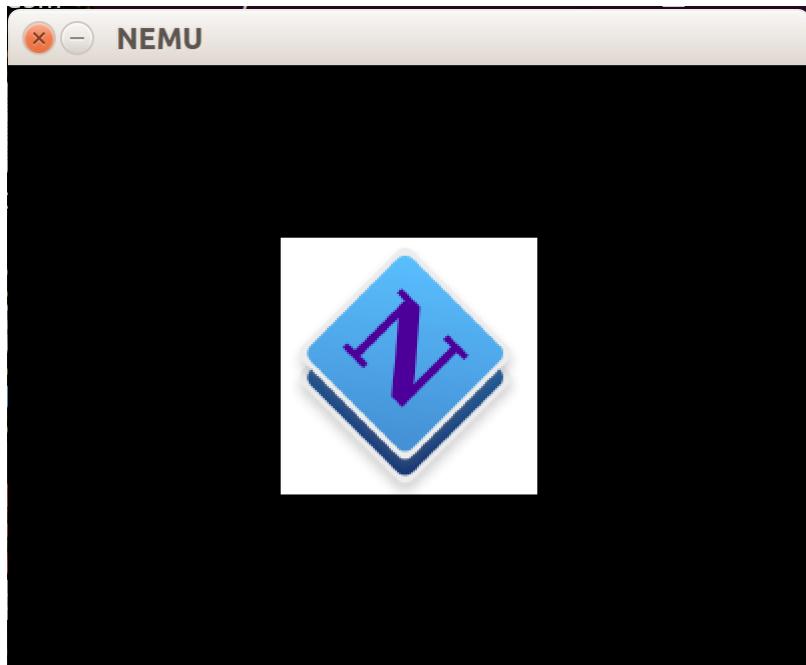
```

由于代码执行效率过低，优化`fb_write()`函数如下。

```

21 extern void getScreen(int* p_width, int* p_height);
22 void fb_write(const void *buf, off_t offset, size_t len) {
23     assert(offset % 4 == 0 && len % 4 == 0);
24     int index, screen_x, screen_y;
25     int width = 0, height = 0;
26     getScreen(&width, &height);
27
28     for(int i = 0; i < len / 4; i++){
29         index = offset / 4 + i;
30         screen_y = index / width;
31         screen_x = index % width;
32         _draw_rect(buf + i * 4, screen_x, screen_y, 1, 1);
33     }
34 }
```

将main.c中加载的文件修改为`/bin/bmptest`后，执行代码，结果如下。



4.2 代码：把设备输入抽象成文件

把设备输入抽象成文件

你需要在 Nanos-lite 中

- ❖ 实现 `events_read()` (在 `nanos-lite/src/device.c` 中定义), 把事件写入到 `buf` 中, 最长写入 `len` 字节, 然后返回写入的实际长度. 其中按键名已经在字符串数组 `names` 中定义好了. 你需要借助 IOE 的 API 来获得设备的输入.
- ❖ 在文件系统中添加对 `/dev/events` 的支持.

让 `Nanos-lite` 加载 `/bin/events`, 如果实现正确, 你会看到程序输出时间事件的信息, 敲击按键时会输出按键事件的信息.

首先在 `nanos-lite/src/device.c` 下实现按键事件的函数 `events_read()` 如下。

```

11 size_t events_read(void *buf, size_t len) {
12     char str[20];
13     bool down = false;
14     int key = _read_key();
15     if(key & 0x8000){
16         key ^= 0x8000;
17         down = true;
18     }
19     if(key != KEY_NONE)
20         sprintf(str, "%s %s\n", down ? "kd" : "ku", keyname[key]);
21     else
22         sprintf(str,"t %d\n", _uptime());
23     if(strlen(str) <= len){
24         strncpy((char*)buf, str, strlen(str));
25         return strlen(str);
26     }
27     Log("strlen(event) > len, return 0");
28     return 0;
29 }
```

修改fs_read()函数如下。

```

78     if(fd == FD_EVENTS){
79         return events_read(buf, len);
80     }
```

运行代码，结果如下。

```

[srsrc/main.c,19,main] 'Hello World!' from Nanos-lite
[srsrc/main.c,20,main] Build time: 11:10:39, Jun 1 2020
[srsrc/ramdisk.c,26,init_ramdisk] ramdisk info: start = 0x1020c0, end = 0x370aca,
size = 2550282 bytes
[srsrc/main.c,27,main] Initializing interrupt/exception handler...
[srsrc/fs.c,31,init_fs] set FD_FB size=480000
[srsrc/loader.c,17,loader] filename=/bin/events, fd=26
receive event: t 259
receive event: t 483
receive event: t 716
receive event: t 959
receive event: t 1192
receive event: t 1443
receive event: t 1683
receive event: t 1920
receive event: t 2161
receive event: t 2399
receive event: t 2648
receive event: t 2880
```

4.3 运行仙剑奇侠传

运行仙剑奇侠传故事模式需要添加指令cwtl指令，

nemu/src/cpu/exec/data-mov.c中添加该指令执行函数如下。

```

78 make_EHelper(cwtl) {
79     if (decoding.is_operand_size_16) {
80         //TODO();
81         panic("operand size should be 32");
82     }
83     else {
84         //TODO();
85         rtl_lr_w(&t0, R_AX);
86         rtl_sext(&t0, &t0, 2);
87         rtl_sr_l(R_EAX, &t0);
88     }
89     print_asm(decoding.is_operand_size_16 ? "cbtw" : "cwtl");
90 }
91 }
```

补充opcode表如下。

```

113 /* 0x98 */ EX(cwtl), EX(cltd), EMPTY, EMPTY,
```

运行仙剑奇侠传，结果如下。



4.4 必答题

必答题

文件读写的具体过程 仙剑奇侠传中有以下行为：

- ❖ 在 `navy-apps/apps/pal/src/global/global.c` 的 `PAL_LoadGame()` 中通过 `fread()` 读取游戏存档
- ❖ 在 `navy-apps/apps/pal/src/hal/hal.c` 的 `redraw()` 中通过 `NDL_DrawRect()` 更新屏幕

请结合代码解释仙剑奇侠传，库函数，libos, Nanos-lite, AM, NEMU 是如何相互协助，来分别完成游戏存档的读取和屏幕的更新。

参考老师给的实验指导，结合个人理解，作答如下。

在 `navy-apps/apps/pal/src/hal/hal.c` 中，通过调用 `NDL_DrawRect` 函数设置屏幕所需的像素节点，之后调用 `NDL_Render` 进行图像绘制。`NDL_Render` 对显存的抽象文件 `/dev/fb` 执行了 `lseek`、`fwrite` 写操作与 `fflush` 刷新操作来实现屏幕更新。以 `fwrite` 函数为例（`fread` 类同），该库函数调用 `libc` 中的 `write` 函数，最终调用 `libs` 中的 `syscall` 函数，该函数通过 `int 0x80` 的内联汇编语句进行系统调用。之后 `nemu` 执行 `int` 指令，在 `nemu` 中完成异常的硬件处理之后，根据 `idt` 内容切换至 `AM` 的目标地址。在 `AM` 中保存现场，将异常封装成事件，并调用 `nanos-lite` 中的异常函数。

对于显存的抽象文件 `/dev/fb`，`sys_write` 调用 `fs_write` 进行文件系统的操作，因此进入了显存的写函数 `fb_write`，在 `nanos-lite` 中的 `fb_write` 调用 `AM` 中的 `_draw_rect`，从而实现图像绘制。

以上是本次实验第三阶段内容。

以上是本次实验内容，感谢批阅！
