Introduction to Computer System

PA3

实验报告

姓名: 郑奘巍

学号: 171860658

邮箱: zzw@smail.nju.edu.cn

院系: 计算机科学与技术系

2018 年 12 月 4 日

第一部分　实验进度

我完成了所有内容（包括 diff-test 的开关和快照的储存、加载，马里奥和功夫的运行）

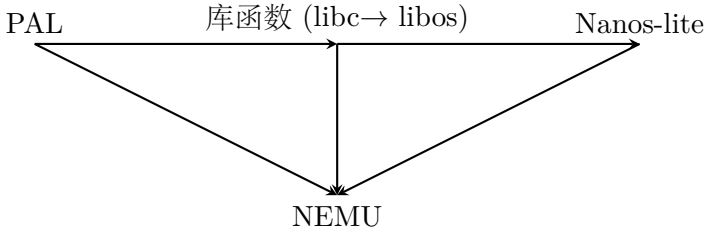
第二部分　必答题

问题 1 *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.*

解答 *(a) 在仙剑奇侠传的 PAL_LoadGame() 函数中,通过 fread(&s, sizeof(SAVEDGAME), 1, fp) 读取存档。其中, fp 是指向传递进来的存档文件名的文件指针, SAVEDGAME 是一个游戏的整个状态，用以恢复游戏。之后游戏状态将通过 s 恢复。*

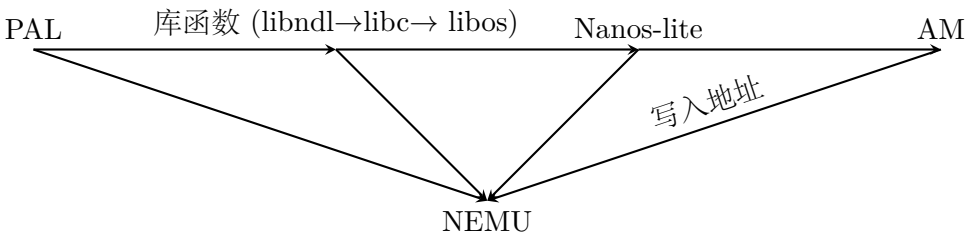
fread 函数属于库函数，处于 libc 的 stdio 库中，直接调用同文件中的 _fread_r。该函数最终调用调用了文件类型的读取函数，读取函数使用了 libos 中的 _read 函数。_read 函数会通过 int $0x80 产生系统调用。int $0x80 会使 eip 跳转到 IDTR 指示的地址，最后将上下文传给 Nanos-lite 的 do_syscall 函数处理系统调用，处理 SYS_read 类型的系统调用。

该类型系统调用使用 fs_read 函数。该函数中，利用 ramdisk_read 读取文件信息。ramdisk_read 中，利用 memcpy 直接拷贝了内存的值。这些操作都是由 NEMU 支持的。



图：fread 的各部分相互协作

(b) redraw 函数将画面信息保存在数组 fb 中，利用 libndl 中的 NDL_DrawRect 绘制图像。该函数中，会调用到 printf,putchar,fwrite 等函数，位于 libc 中，libc 又会调用 libos 产生系统调用。由于写入的是虚拟文件/dev/fb, 写入函数会调用 AM 提供的 draw_rect 函数，draw_rect 函数又是通过写入被映射为显存的地址控制显示。



图：NDL_DrawRect 的各部分相互协作

第三部分　选做题

Diff-test 的开关　设置一全局变量控制 diff-test 开关，当 detach 后，difftest_step(), difftest_skip_dut() 和 difftest_skip_ref() 直接返回。

对于 attach 操作，我们还要将栈区、寄存器和 IDTR 赋值。具体操作如下：

```
void difftest_attach(){
    ref_difftest_memcpy_from_dut(0, guest_to_host(0), 0x7c00);
    ref_difftest_memcpy_from_dut(ENTRY_START, guest_to_host(ENTRY_START), memsize);
    ref_difftest_memcpy_from_dut(0x7e00, &cpu.idtr.size, 2);
    ref_difftest_memcpy_from_dut(0x7e02, &cpu.idtr.addr, 4);
    uint8_t inst[] = {0x0f,0x01,0x1D,0x00,0x7e,0x00,0x00};
    ref_difftest_memcpy_from_dut(0x7e40, inst, sizeof(inst));
    CPU_state tmp = cpu;
    tmp.eip = 0x7e40;
    ref_difftest_setregs(&tmp);
    ref_difftest_exec(1);

    ref_difftest_setregs(&cpu);
}
```

图：attach 还原操作

快照　save 操作中我们依次打印寄存器值和所有内存的值，load 操作中我们根据打印的次序读出并还原即可。具体操作如下：

```
static int cmd_save(char *args){
    if (args==NULL){
        puts("Path must be specified!");
        return 0;
    }
    char *path;
    path = strtok(NULL, " ");
    FILE *fp = fopen(path, "w");
    assert(fp);
    fwrite(&cpu, 4, 10, fp);
    fwrite(pmem, 1, PMEM_SIZE, fp);
    printf("Snapshot saved to %s\n",path);
    fclose(fp);
    return 0;
}
```

图：save 操作

```
static int cmd_load(char *args){
    if (args==NULL){
        puts("Path must be specified!");
        return 0;
    }
    char *path;
    path = strtok(NULL, " ");
    FILE *fp = fopen(path, "r");
    assert(fp);
    fread(&cpu, 4, 10, fp);
    fread(pmem, 1, PMEM_SIZE, fp);
    printf("Snapshot loaded from %s\n",path);
    fclose(fp);
    return 0;
}
```

图：load 操作

函数参数传递　为了运行可以让 LiteNES 选择运行马里奥或功夫，需要让运行时能够传递参数。由于原来的字符串保存处会被新加载的程序覆盖，需要新增保存的空间。具体代码如下：

```c
void args_uload(PCB *pcb, const char *filename, char *argv[], char *envp[]){
    Log("\n\n\nHERE\n\n\n");
    char sargv[16][32];
    char senvp[16][32];
    char *pargv[16];
    char *penvp[16];

    int i = 0;
    while (argv[i]){
        strcpy(sargv[i], argv[i]);
        pargv[i] = sargv[i];
        i++;
    }
    int argc = i;

    i = 0;
    while (envp[i]){
        strcpy(senvp[i], envp[i]);
        penvp[i] = senvp[i];
        i++;
    }

    uintptr_t entry = loader(pcb, filename);

    ((void(*)(int, char **, char *[]))entry) (argc, pargv, penvp);
}
```

图：LiteNES 的参数传递

第四部分　遇到的问题

打开文件和关闭文件时未清空 open_offset　这个问题是整个 PA3 中卡了我最长时间的问题。头一次发现是仙剑奇侠传中无法读取存档，几经 DEBUG 我在关闭文件时清空了偏移量。但在最后 LiteNES 返回选择新程序时，未经选择即跳转到上次选择的程序中。这是打开文件时未清空偏移量所置。