PA3 实验报告

匡亚明学院 赵超懿 191870271

实验进度：

我完成了PA3中所有必做内容，关于蓝框选做，我仅完成了使画布位于屏幕中央。

必做题：

1. 理解上下文结构体的前世今生 (见PA3.1阶段)

   答：这个问题我再解决时主要从指令的执行角度来解决。

   在保存上下文的过程中，cpu执行了int 指令后，进入到了raise_intr函数，在这个函数中，cpu按照顺序，先后将eflags，cs，ret_addr压入栈中，然后通过idtr寄存器计算出跳转的地址，然后下面跳转到__ am_vectrap函数，向栈中压入 0x81(int指令的值) _am_asm_trap函数，在这个函数中，调用了pusha和push 0，然后是esp寄存器，然后进入到call __am_irq_handle函数，将使用栈存储的数据，即context。

   所以在context结构体中的，在栈中从高地址到低地址依次为 eflags, cs ,ret_addr , 0x81,所有通用寄存器以及0和esp，所以只需要在**Context**结构体中将以上**对应的值倒过来按顺序**写在Context中即可。在这个结构体中，void *cr3 指针应该指向push的0，其余对应相应的值。

   对于是如何组织起来的，在于数据在栈上的组织，最后push进入的esp是作为传入的Context*指针参数，其指向的结构体实体便是在栈上的压入的数据。指针参数于是传入了__am_irq_handle 函数。

2. 理解穿越时空的旅程 (见PA3.1阶段)

   答：yield()函数调用后，会执行int 81指令，然后按照在第一个必做题中所说：

   进入到了raise_intr函数，在这个函数中，cpu按照顺序，先后将eflags，cs，ret_addr压入栈中，然后通过idtr寄存器计算出跳转的地址，然后下面跳转到__ am_vectrap函数，向栈中压入 0x81(int指令的值) _am_asm_trap函数，在这个函数中，调用了pusha和push 0，然后是esp寄存器，然后进入到call __am_irq_handle函数。

在__am_irq_handle 函数中，然后识别异常号81，将事件赋值为EVENT_YIELD, 然后进行事件处理，根据在cte_init中注册的事件来进行do_event进行事件分发，事件分发识别之前赋值的事件，判断事件为EVENT_YIELD后，输出一段话，然后开始返回，恢复上下文，程序返回到 _am_asm_trap 调用 _am_irq_handle之后的敌方，将栈指针esp增加到 pusha指令执行后的地方，然后将之前保存的通用寄存器全部pop到寄存器中，然后执行iret指令，将之前push进栈中的pc，cs寄存器以及eflags ，都pop到对应的寄存器中中，然后在yield()函数调用int 81后的位置返回，恢复到程序调用yield前的状态。

3. hello程序是什么, 它从而何来, 要到哪里去 (见PA3.2阶段)

   我们知道`navy-apps/tests/hello/hello.c`只是一个C源文件, 它会被编译链接成一个ELF文件. 那么, hello程序一开始在哪里? 它是怎么出现内存中的? 为什么会出现在目前的内存位置? 它的第一条指令在哪里? 究竟是怎么执行到它的第一条指令的? hello程序在不断地打印字符串, 每一个字符又是经历了什么才会最终出现在终端上?

   答：hello程序被编译后，其ELF文件处于二进制文件的开头。Hello文件处于ramdisk.img文件的开头。

   nanos-lite中的init_proc函数调用naive_uload函数加载文件。最后调用loader.c文件加载文件。首先将大小为sizeof(Elf_Ehdr)的文件读入，根据elf头文件的格式，可以获得program header的数量，是否加载以及在文件中偏移量，于是loader.c根据这些信息读取program header文件偏移量，文件大小，内存大小，加载信息，虚拟地址，将文件读入到nemu的内存之中，而且elf header中有一个程序进入地址，e_entry，这样系统便可以将文件pc设置为loader的返回值，然后跳转，体现在指令上变为call *%ebx，于是就进入到了文件中的_start的位置（因为elf文件的e_entry指向这个位置），于是nemu开始执行客户程序hello。

   hello的执行过程中，首先调用的是write函数，先后调用 *write_r和 _write函数， _write函数即为libos中的 _write函数，开始系统调用，准备好参数，然后执行int 80指令，该指令属于系统调用事件，于是nanos-lite开始系统调用。这个过程体现在c的代码上为从navy-apps中的syscall.c文件中 _write 调用 _syscall*函数，从这里执行int 80 指令开始系统调用，系统调用识别为sys_write调用，然后交与函数处理，由于输出是stdout ，所以直接使用串口将字符输出。

   之后的printf输出，（此处解释使用了_sbrk的情况），printf申请内存作为缓冲区，直到遇到\n输出。在要输出的过程中，库函数将输出行为转换为系统调用（到write），类似于write的行为，仍然使用fd为stdout的为sys_write的系统调用，使用串口将文件输出出来。然后循环这个过程。

4. 仙剑奇侠传究竟如何运行 运行仙剑奇侠传时会播放启动动画, 动画中仙鹤在群山中飞过. 这一动画是通过`navy-apps/apps/pal/repo/src/main.c`中的`PAL_SplashScreen()`函数播放的. 阅读这一函数, 可以得知仙鹤的像素信息存放在数据文件`mgo.mkf`中. 请回答以下问题: 库函数, libos, Nanos-lite, AM, NEMU是如何相互协助, 来帮助仙剑奇侠传的代码从`mgo.mkf`文件中读出仙鹤的像素信息, 并且更新到屏幕上? 换一种PA的经典问法: 这个过程究竟经历了些什么?

   答：

```
166     //
167     gpGlobals->f.fpFBP = UTIL_OpenRequiredFile("fbp.mkf");
168     gpGlobals->f.fpMGO = UTIL_OpenRequiredFile("mgo.mkf");
169     gpGlobals->f.fpBALL = UTIL_OpenRequiredFile("ball.mkf");
170     gpGlobals->f.fpDATA = UTIL_OpenRequiredFile("data.mkf");
171     gpGlobals->f.fpF = UTIL_OpenRequiredFile("f.mkf");
172     gpGlobals->f.fpFIRE = UTIL_OpenRequiredFile("fire.mkf");
173     gpGlobals->f.fpRGM = UTIL_OpenRequiredFile("rgm.mkf");
174     gpGlobals->f.fpSSS = UTIL_OpenRequiredFile("sss.mkf");
175
```

   从这里加载文件，之间有很多定义的函数，最终会到这里使用fread函数

```
558     //
559     FILE *fp = UTIL_OpenFileAtPath(gConfig.pszSavePath, PAL_va(1, "%d.rpg", iSaveSlot));
560     //
561     // Read all data from the file and close.
562     //
563     size_t n = fp ? fread(s, 1, size, fp) : 0;      Zihao Yu, 2 months ago • common: merge som
564
```

fread是库函数，会产生系统调用SYS_read，这样库函数就和nanos-lite交互，系统调用会使用我所写的fs_read读取文件，最后会成功将数据读取出来。这使用系统调用读取数据的过程中，会使用AM提供的基本环境，如memcpy等，最后这些行为都被编译成为指令，在nemu上运行，这个系统在读取数据的过程中达到了协同。

在pal显示动画的过程中，

```
PAL_ProcessEvent();
dwTime = SDL_GetTicks() - dwBeginTime;

//
// Set the palette
//
if (dwTime < 15000)
{
    for (i = 0; i < 256; i++)
    {
        rgCurrentPalette[i].r = (BYTE)(palette[i].r * dwTime /
        rgCurrentPalette[i].g = (BYTE)(palette[i].g * dwTime /
        rgCurrentPalette[i].b = (BYTE)(palette[i].b * dwTime /
    }
}

VIDEO_SetPalette(rgCurrentPalette);
VIDEO_UpdateSurfacePalette(lpBitmapDown);
VIDEO_UpdateSurfacePalette(lpBitmapUp);

//      Zihao Yu, 2 months ago • common: merge some macros
```

pal使用了SDL库中的函数（VIDEO_其实是SDL库函数）这些函数会使用在libs中miniSDL中的完成的SDL_Update ,SetPalette等函数，而这些函数又是通过NDL函数来实现，NDL中的函数又使用了fwrite等，触发**系统调用，在系统调用中通过AM提供的屏幕，键盘，时钟接口(io接口)（io_write以及io_read）来将画面等信息传递给硬件以及从硬件获取信息，**于是AM，nanos以及AM就协同了起来，最后这些用户程序运行在硬件nemu上，完成开始动画的显示。

实验过程以及实验心得（包括一些我理解的蓝框题的回答）：

总体体会：

在做pa3的3.1和3.2时，觉得难度一般，以为后面的pa大概就这样可以轻松的完成，但是pa3.3给了我致命一击。太难了！

PA 3.1

首先是pa3.1的完成过程：按照讲义实现寄存器，看手册实现行为，然后卡在了 上下文结构体。

上下文结构体（蓝框）：诡异的x86代码：我认为push 的esp的作为事件分发函数的输入参数，指向的是之前压栈的数据。

在这里我犯了一个很傻的错误，没注意到x86要去除一个push 0，于是发现怎么都不对。（！！！），其实这块也不难，就是找到栈中间中值的对应顺序，我看到这块就直到要这样做，但是没看到要改框架代码，于是白费两小时。

PA 3.2

然后是pa3.2的完成过程：首先是loader.c，开始就卡住了，发现加载的文件的大小都是0，（中间明白了二进制文件的复制（一定不是打开vim复制=:=））于是卡住好长时间了（看了makefile明白了update的作用，为以后也节约了一点时间）。最后是请教李晗同学解决了问题，在此感谢李晗同学的帮助。在李晗同学的指导下，我一点点查看编译信息，最后找到真相，是make的过程中需要resouce.S文件被更新，才能够是ramdisk.img文件被编译进去。在这个过程中，我对make的机制有了更加细致的理解。最后再次感谢李晗同学。

蓝框的回答：为什么要置0：这一块是用于全局变量和静态变量的初始化（开始我写的memcpy，后面发现不对才明白的（因祸得福！））

解决了loader，加载文件还是比较简单的，man elf，直接就按照要求将程序段加载。中间使用printf调试，很容易就能解决问题。在loader之后，直至pa3.2结束，都是比较顺利（指没有大问题）的，小问题还是很多的。

在完成系统调用时还是遇到一点问题的，主要是返回值的问题，比如忘了设置返回值，出现和讲义描述现象不相符的情况。不过最后还是很快解决了。最后的堆区管理，实现起来也是很简单，也就不贴代码了。

3.2的最后是堆区管理，**重点：这里没有遇到问题，后面出现问题后找来找去还是找到了这里，然后发现又不是这里的问题（太折磨了！！！）**

当然在3.2阶段还有最大的收获：**在这个阶段写系统调用的过程中，我突然就对整个PA有了很清晰的认识，从硬件到AM到naons以及navy，对于其整个大的工作机制基本都清晰了（之前一直有些模糊，不是很明白，现在总算是明白了整个系统的工作原理了！！！）**

PA 3.3 !!!

PA3的噩梦，感觉整个人变成了GDB。每天都在debug。基本要写的代码都重写过两次（基本全部重写），除了SDL_Pollevent和SDL_Waitevent和文件系统代码。（哭了：满怀信心开始pa3，结果整个人都裂开）

首先是文件系统：这个其实还好，查一查手册，了解下行为。基本就解决了，然后是重写loader，这个过程按理说应该是比较简单的，但是我不直到为什么就搞不对，留下了非常多的调试信息：

```
static uintptr_t loader(PCB *pcb, const char *filename) {
  //TODO();
  //uint32_t size = get_ramdisk_size();
  //printf("%d\n",size);
  Elf_Ehdr elfhdr;
  Elf_Phdr prohdr;
  //printf("%s\n",filename);
  size_t fd = fs_open(filename,0,0);
  //printf("%d\n",fd);
  fs_read(fd,&elfhdr,sizeof(Elf_Ehdr));
  //printf("%x %x\n",elfhdr.e_phoff,elfhdr.e_phnum);
  assert(fd != -1);
  //printf("%x %x %x\n",elfhdr.e_entry,elfhdr.e_phentsize,elfhdr.e_ehsize);
  for(int i = 0;i < elfhdr.e_phnum;i++)
  {
    fs_lseek(fd,elfhdr.e_phoff+i*sizeof(Elf_Phdr),SEEK_SET);
    fs_read(fd,&prohdr,sizeof(Elf_Phdr));
    if(prohdr.p_type == PT_LOAD){
      fs_lseek(fd,prohdr.p_offset,SEEK_SET);
      fs_read(fd,(void *)prohdr.p_vaddr,prohdr.p_filesz);
      //printf("from %x %x size = %x \n",prohdr.p_vaddr,prohdr.p_offset,prohdr.p_filesz);
      memset((void*)(prohdr.p_vaddr+prohdr.p_filesz),0,prohdr.p_memsz-prohdr.p_filesz);
    }//之前用的memcpy，不愧是我

  }
  fd = fs_close(fd);
  assert(fd == 0);
  printf("%s File Loaded\n",filename);
  return elfhdr.e_entry;
}
//静态变量的初始化问题，好像是在programmmer_header中加载的，
```

可以看到有很多调试信息，后面的文件基本也都是这样过去的。（调试的代码基本和代码差不多一样了，这还是删掉了一些的情况下，不过调试理论还是很好用的）。

操作系统上的IOE，如果在PA2没有理解整个PA的工作机制，那么我可能还要花时间去理解，但是在这里，我已经明悟了，所以写起来很快（仅仅是写555），串口和时钟还是比较简单的，然后在仅在此处是屏幕正常工作也是比较简单的。对的，仅在此处，这也是为什么我后来改了很多次的原因。

蓝框选做：在理解画布概念后实现，不过在第一次写并没有写。

下面就到了错误时刻：（先跳过顶点算数，后面再讲，快进到APP）

1. nslider，一次成功：为什么再第一次实现有问题的基础上可以对呢？因为恰好全屏（555）
2. menu 也是一次成功：成功原因同上：全屏
3. nterm是第一个转折点：首先：在实现NDL时屏幕我第一次用了sprintf，在用这个的时候我还考虑到要以%c来进行输出。在第一次运行nterm时，我发现结果是满屏的白色，然后我阅读了源码，发现结果不应该是满屏，然后就修改了opencanvas函数，修改好后，可以正常显示了，然后发现帧率过低，然后开启了第一次重构，将sprintf修改为fwrite，然后发现性能大幅度提升。

第一阶段到此结束，似乎还挺顺利。（我当时也这么觉得），然后后来就不是了，当开始运行仙剑奇侠传时，一切埋下的坑就出现了（不仅仅是软件，还有硬件，我裂开来）

第二阶段：首先是nanos的问题，首先，由于对手册理解不正确，pal直接黑屏，然后我去看自己的源码，发现对画布的理解有问题，然后从NDL_Draw到SDL基本和画布相关的都重新写了一遍，然后先重复之前的测试，然后在前面的就直接挂掉，好，我继续bug，再次走到pal面前，此时，我的video实现基本都还可以（还是有一些小问题的），然后出现了非常神奇的错误，PAL的启动画面能否成功变成随机的，想了一段时间意识到和随机有关系的只有时间了，啪的一查，很快啊，果然是时间错了（除法变成了取模），我看着我之前犯下的xx错误，觉得我是个x（不是）。好了现在可以正常启动pal，果然不再是随机事件了。第二阶段结束。

然后就到了第三阶段：进去，选项上面有黑色阴影，此时我甚至想着是不是因为我们使用了RGB而没有用a的值导致的，事实证明，机器永远是对的，只要有错那么一定是我（太真实了），为了这个错误，我阅读了很多仙剑的源码（看这种规模的文件好痛苦，而且重点还有我的vscode有些好用的功能无法对pal生效，我又裂开）。然后慢慢看源码，然后打断点，输出调试，一上午过去才修好这个bug，果然还是写SDL对画布的理解不够，好，再次重写（小规模重写），修好了我都要泪目了，然后x86跑有些，进去就发现两个问题，首先打不了怪，然后一遇到有任务对话，直接错误，我直接懵逼(55555555)，然后调了很长时间，没有进展。后来还发现了堆区一直增长的问题，（上面提到过，再次裂开）

我周日一天就做了这么一点事。当然，还学会了用navy作为基础设施。但是还没有彻底定位到错误（那天晚上才学会用navy作为基础设施的）。然后我偶然发现了一些问题，堆区还在不停增长——sbrk出问题了，但是还是不会解决。

终于要到结局了：（太难了）

首先是堆区问题，当发现即使不动程序的情况下，他仍然在增长，然后看了1个多小时没想法，只发现了是fopen在不停申请，然后我去洗澡，在洗澡时我突然明悟了NDL_init的作用，就是设置一些全局变量的指针，在init时对其初始化，这样就不会一直打开了（Yes！）

然后是pal运行遇到对话直接退出的问题，我在学会了native这个强而有力的工具后很快确信是我的硬件层nemu出bug了，我就很绝望，然后想怎么办呢？先往下看，看到difftest的开关，有希望，然后上手一做，各种出问题（有些难），然后就卡住了。最后是在方宇航同学的帮助下进行替换测试（仅nemu），终于定位到了nemu的错误是eflags的设置，然后解决，进行pa2中进行的一系列测试，然后跑跑仙剑，大功告成（完结撒花）。

收获：首先是实验课刚讲的调试理论，有了很大的作用，我打端点的能力大幅度提升，在后期深陷bug中时也能很快找到bug，然后是替换测试（也是上课讲的），同样使用了类似调试理论的方法，二分快速定位到bug。

感受：

机器真的永远是对的！！！

即使经过充分的测试，依然很有可能出现bug，在我确信是我的nemu'出现问题是，我很难相信，因为在pa2中，我的nemu可以运行讲义要求的所有指令，并且带difftest依然可以通过所有的测试。所以，经过测试的代码依然要**谨慎对待**。

关于测试，定点数也是需要测试的（5555），之前写过bug导致pal战斗不正常。也验证了为经过测试的代码都是不正确的。

对以后的启示：在写pa3的过程中我的方法确实不好，经常出现对手册理解不到位，观点不全面的原因，在SDL中体现的更加明显，写有一定规模的代码一定**要从全局出发，规划好理解到位再写（当然边看边写容易增加理解）**。另一个方面就是工具了，vscode的使用在pa3中给了我很大的帮助（之前花了一些时间在学习vscode是值得的。）

最后，我也很高兴，在最后深陷bug长达1周的时间中，我没有放弃。在pa3的整个过程中，除了两位同学的帮助，我的其他内容均独立完成，自己查看手册，自己找bug，自己修好，写好后又重构，一遍又一遍，体会到的写一份好的代码的困难，**在这个过程中，我的能力也得到大幅度提升。**回首pa3，还是一份愉快的旅程。

实验报告在pa3仙剑奇侠传成功运行后完成，有很多细节遗漏，以后应该注意，应该及时记录问题，跟着pa的进度写实验报告。

最后的截图的展示：

注释与代码的交错（写了很多遍，也调试了很多），纪念不易的PA3。

```
237  void SDL_SetPalette(SDL_Surface *s, int flags, SDL_Color *colors, int firstcolor, int ncolors) {
238    assert(s);
239    assert(s->format);
240    assert(s->format->palette);
241    assert(firstcolor == 0);
242
243    s->format->palette->ncolors = ncolors;
244    memcpy(s->format->palette->colors, colors, sizeof(SDL_Color) * ncolors);
245    /* for(int i = 0;i < ncolors;i++)
246        printf("%dth r = %x g = %x b = %x\n",i,colors[i].r,colors[i].g,colors[i].b); */
247    //printf("%d\n",s->flags);
248    if(s->flags & SDL_HWSURFACE) {
249      assert(ncolors == 256);
250      /* for(int i = 0;i < ncolors;i++)
251        printf("%dth r = %x g = %x b = %x\n",i,colors[i].r,colors[i].g,colors[i].b); */
252      /*printf("\nnew\n");
253      for(int i = 0;i < s->w;i++)
254        printf("%dth  color = %x\n",i,colors[s->pixels[i]]); */
255      for (int i = 0; i < ncolors; i ++) {
256        uint8_t r = colors[i].r;
257        uint8_t g = colors[i].g;
258        uint8_t b = colors[i].b;
259        //printf("ith = %d r = %x g = %x b = %x\n",r,g,b);
260      }
261      /* for(int i = 0; i< s->w*s->h;i ++)
262        printf("place %d   idx = %d color = %x\n",i,s->pixels[i],colors[s->pixels[i]]); */
263      SDL_UpdateRect(s, 0, 0, 0, 0);
264      //while(1);s      You, seconds ago • Uncommitted changes
265    }
```

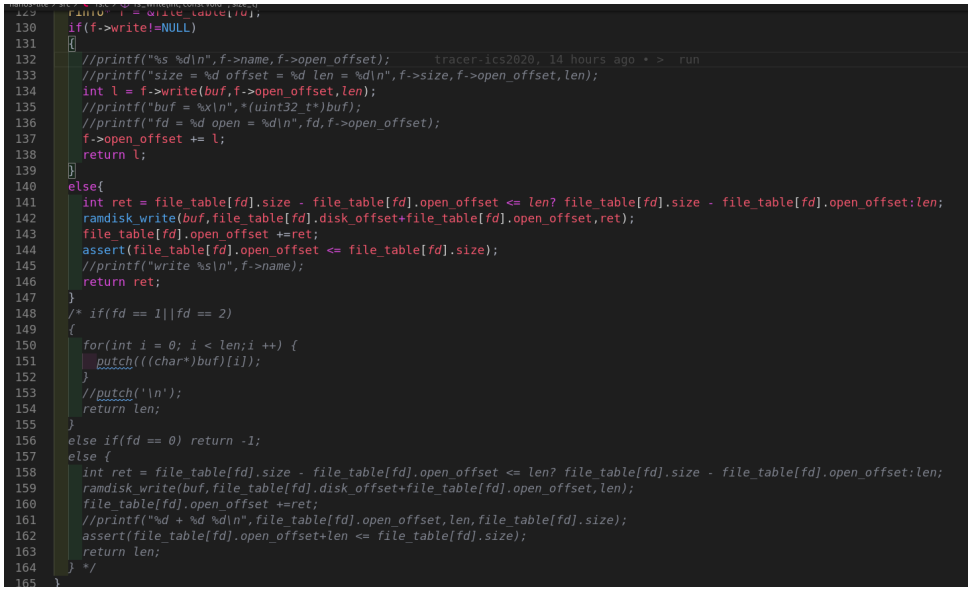
```
  uint8_t* dst_color = dst->pixels,*src_color = src->pixels;
  uint32_t color_width = dst->format->palette?1:4;
  //printf("color width %d\n",color_width);
  for(int i = 0;i < src_h;i++)
    memcpy(dst_color+color_width*((i+dst_y)*dst->w+dst_x),src_color+color_width*((i+src_y)*src->w+src_x),color_width*src_w);
    //memcpy(dst_color+4*(i+dst_y)*dst->w+4*dst_x,src_color+4*(i+src_y)*src->w+4*src_x,4*src_w);
  /* for(int i = 0; i < h;i++)
    //memcpy(dst_color+dst_w*(y+i)+x,src_color+i*w,4*w);
    for(int j = 0;j < w;j ++)
    {
      dst_color[dst_w*(y+i)+x+j] = src_color[i*w+j];
    } */

  //printf("please implement me\n");
  //assert(0);
}
```

```
  for(int i = 0;i < h;i ++)
    for(int j = 0;j < w;j ++)
    {
      canvas[(y+i)*canvas_w+x+j] = pixels[i*w+j];
    }
  for(int i = 0;i < canvas_h;i ++)
  {
    //printf("seek %d\n",4*((i+place_y)*screen_w+place_x));
    fseek(fb,4*((i+place_y)*screen_w+place_x),SEEK_SET);
    fwrite((void*)(canvas+i*canvas_w),1,4*canvas_w,fb);
  }
  //printf("NDL %p %d\n",fb_sync,ftell(fb_sync));
  fseek(fb_sync,0,SEEK_SET);
  fwrite("1",1,1,fb_sync);
  //printf("refresh %d\n",k++);
  /* for(int i = 0;i < h;i++)
  {
    fseek(fp,4*((y+i)*screen_w+y),SEEK_SET);
    fwrite((void*)pixels,sizeof(uint32_t),w,fp);
    //printf("seek %d\n",4*((y+i)*wid+y));
    for(int j = 0;j < 4*w;j++)
    {
      fwrite((void*)pixels,sizeof(uint32_t),h,fp);
      fprintf(fp,"%c",((char*)pixels)[4*w*i+j]);
      //printf("p = %d %d\n",w*i+j,pixels[i*w+j]);
    }
  }*/
  /* printf("%p %d\n",fb_sync,ftell(fb_sync));
  fseek(fb_sync,0,SEEK_SET);
  sprintf(fb_sync,"%c",'1'); */
  //fwrite(buf,1,1,fb_sync);
```

```
  Finfo* f = &file_table[fd];
  if(f->read == NULL)
  {
    if(f->open_offset >= f->size)
    {
      assert(f->open_offset <= f->size);
      //printf("%s open_offset = %d size = %d FULL\n",f->name,f->open_offset,f->size);
      return 0;
    }
    else {                                                    size_t <unnamed>::open_offset
      size_t l = len <= file_table[fd].size - file_table[fd].open_offset? len:file_table[fd].size - file_table[fd].open_offset;
      //printf("third %d %d %d\n",file_table[fd].open_offset,file_table[fd].size,l);
      ramdisk_read(buf,file_table[fd].disk_offset+file_table[fd].open_offset,l);
      file_table[fd].open_offset = file_table[fd].open_offset+l;
      assert(file_table[fd].open_offset <= file_table[fd].size);
      return l;
    }
  }
  else{
    //printf("%d\n",len);
    int ret = f->read(buf,file_table[fd].open_offset,len);
    f->open_offset+=len;
    //size_t l = len <= file_table[fd].size - file_table[fd].open_offset? len:file_table[fd].size - file_table[fd].open_offset;
    //file_table[fd].open_offset = file_table[fd].open_offset+l;
    return ret;
  }//以后可能出现问题

}//remained to be thinking about fd == 0,1,2, .e.t stdin,stdout,stderror
  // else if(file_table[fd].open_offset+len > file_table[fd].size)
  // {
  //   printf("sec2 %d %d %d\n",file_table[fd].open_offset,file_table[fd].size,len);
  //   ramdisk_read(buf,file_table[fd].disk_offset+file_table[fd].open_offset,file_table[fd].size-file_table[fd].open_offset);
  //   file_table[fd].open_offset = file_table[fd].size;
  //   printf("%d\n",);
  //   return file_table[fd].size-file_table[fd].open_offset;
  // }
```



```
  Finfo* f = &file_table[fd];
  if(f->write!=NULL)
  {
    //printf("%s %d\n",f->name,f->open_offset);      tracer-ics2020, 14 hours ago • >  run
    //printf("size = %d offset = %d len = %d\n",f->size,f->open_offset,len);
    int l = f->write(buf,f->open_offset,len);
    //printf("buf = %x\n",*(uint32_t*)buf);
    //printf("fd = %d open = %d\n",fd,f->open_offset);
    f->open_offset += l;
    return l;
  }
  else{
    int ret = file_table[fd].size - file_table[fd].open_offset <= len? file_table[fd].size - file_table[fd].open_offset:len;
    ramdisk_write(buf,file_table[fd].disk_offset+file_table[fd].open_offset,ret);
    file_table[fd].open_offset +=ret;
    assert(file_table[fd].open_offset <= file_table[fd].size);
    //printf("write %s\n",f->name);
    return ret;
  }
  /* if(fd == 1||fd == 2)
  {
    for(int i = 0; i < len;i ++) {
      putch(((char*)buf)[i]);
    }
    //putch('\n');
    return len;
  }
  else if(fd == 0) return -1;
  else {
    int ret = file_table[fd].size - file_table[fd].open_offset <= len? file_table[fd].size - file_table[fd].open_offset:len;
    ramdisk_write(buf,file_table[fd].disk_offset+file_table[fd].open_offset,len);
    file_table[fd].open_offset +=ret;
    //printf("%d + %d %d\n",file_table[fd].open_offset,len,file_table[fd].size);
    assert(file_table[fd].open_offset+len <= file_table[fd].size);
    return len;
  } */
}
```

感谢

感谢方宇航同学在我在运行pal时解决nemu硬件 bug中提供的帮助。

感谢李晗同学的答疑解惑。

衷心感谢两位同学！