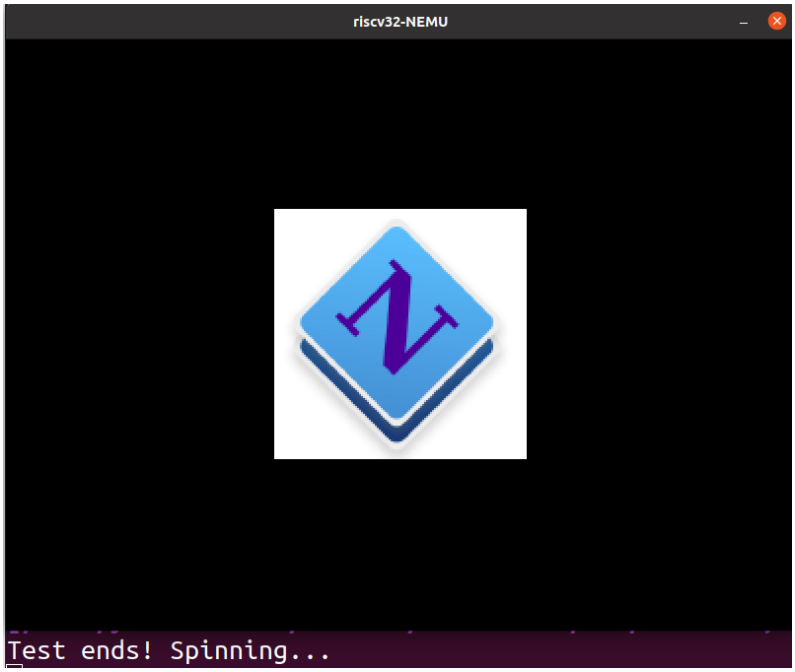
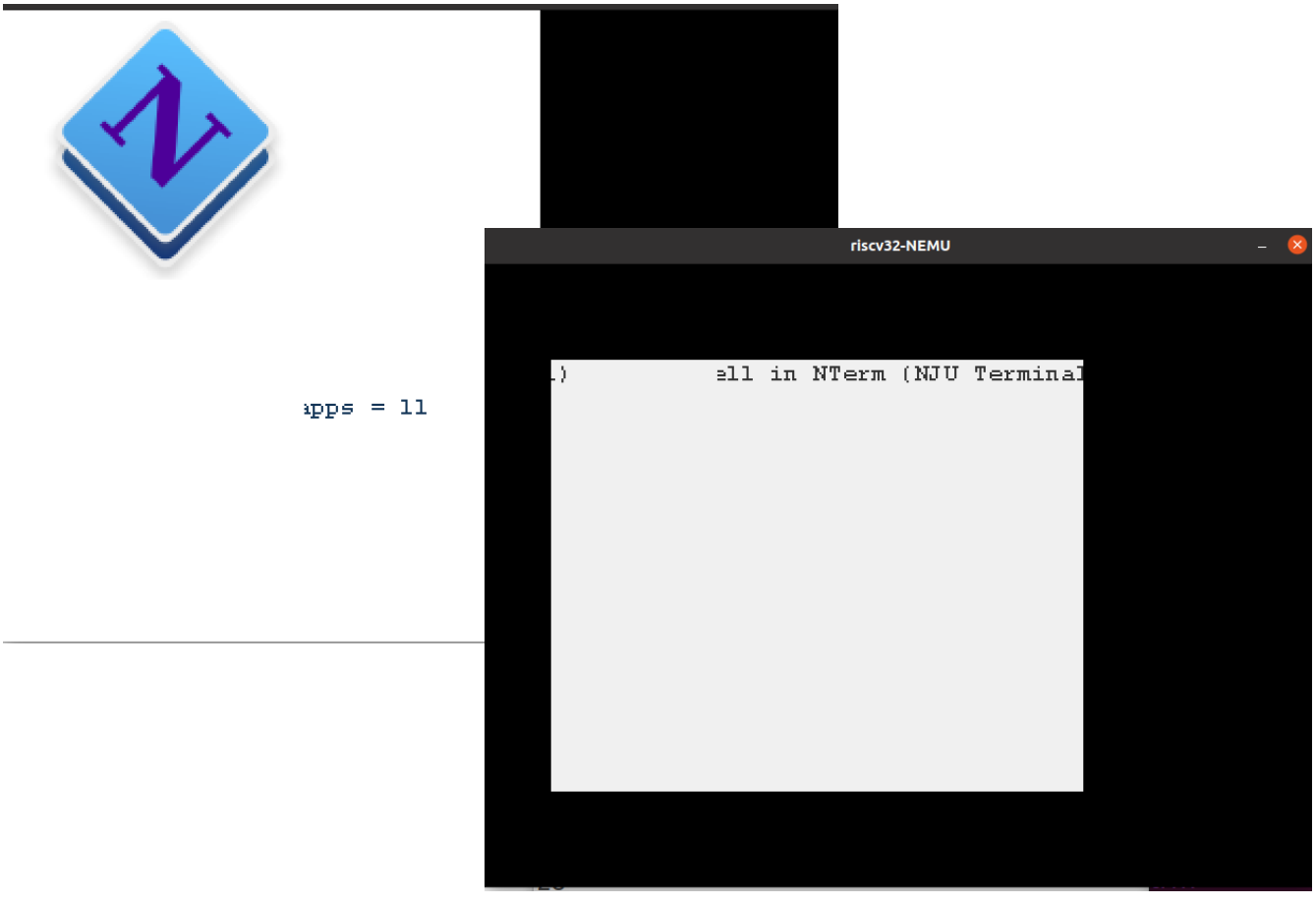
实验进度：我已完成 PA3.1 至 3.2 所有内容，以及 3.3 的部分内容，并通过 OJ 的自陷测试。3.3 进度一半，至“NJU Terminal”部分，nterm 和 menu 的绘图 bug 还未全部解决。

nanos-lite 的 VGA 正确实现截图：



menu 和 nterm 进度（bug 暂时没有 de 完）：



必答题：

理解上下文结构体的前世今生：

c 指向的上下文结构是 am 中定义的 Context，对 riscv32 来说位于头文件 riscv32-nemu.h 中，具体成员为 32 个通用寄存器和三个系统相关寄存器 mcause、mstatus 和 mepc。这个结构体可用于从软件层面访问寄存器，并作为 am 处理中断和异常的渠道。各寄存器内容随指令操作与 nemu 中寄存器值保持一致；mcause 用于判断事件的异常号，mepc 则是保存从异常处理过程返回的时候继续执行的位置，二者均在发生异常或是中断的时候被赋值；mstatus 存放处理器的状态，从一开始就会被初始化。

trap.S 中保存了异常入口。实现的新指令中的 csrw 可以将该入口保存至新增 mtvec 寄存器中，用于异常或是中断时跳转至设置好的位置。同时 trap.S 中的汇编也会依次对 Context 的各个成员进行调整，因此需要将该结构体重新组织以正确设置异常入口。

理解穿越时空的旅程：

yield()中包含两条内联汇编代码：第一条将-1 保存至 a7 寄存器，第二条则使用 ecall 指令向系统发出异常。ecall 指令利用 isa_raise_intr 函数在硬件中保存 pc 和异常号，最后跳转至异常处理入口（由 cpu.mtvec 保存）。通过 a7（GPR1）寄存器为-1 可以识别出该异常为 EVENT_YIELD，因此在 am 中 ev.event 参数被置 1。接下来把 ev 和上下文交由 user_handler 函数处理。do_event 函数中进入 yield 事件分支，完成处理后通过 mret 指令回到 cpu.mepc 处继续执行。

hello 程序是什么，它从何而来，要到哪里去：

hello 程序通过人为的复制被装入 ramdisk 镜像之中，然后由 loader 函数首先调用 ramdisk_read 函数读取 elf 头，因为其中只有一个程序，所以访问第一个字节就能读取 elf 头，偏移量为 0。根据 elf 头获取程序头表的偏移量和表项数目后，将程序头表一项项读入、并检查其 p_type，若为 PT_LOAD 则利用 ramdisk_read 函数读取其以偏移量 offset 为起点，大小为 filesz 的内容至其指定的 vaddr 处，再将[vaddr + filesz, vaddr + memsz)这段内存用 memset 函数置为 0。

这是由于 bss 节中存在一些未初始化的全局变量等信息，存储时不必为他们分配空间，但是在程序加载时需要为他们留下内存空间，不清 0 会导致未初始化全局变量被错误赋值、或者出现其他异常。

将可装入段加载到内存中指定位置后，将 loader 函数 elf.entry 返回，由主程序调用跳转到程序的 entry 处，即第一条指令的首地址，开始执行用户程序。

hello 程序不断打印字符串，每一个 printf 首先通过 brk 系统调用申请堆区存放格式化字符串。brk 的系统调用号为 9，保存在 a7 中，有效参数为 program_break + increment，保存在 a0 中，同样经过 ecall 利用 isa_raise_intr 函数在硬件中保存 pc 和异常号，最后跳转至异常处理入口（由 cpu.mtvec 保存）。通过 a7（GPR1）寄存器为 9 可以识别出该异常为 EVENT_SYS，因此在 am 中 ev.event 参数被置 2。接下来把 ev 和上下文交由 user_handler 函数处理。do_event 函数中进入 syscall 事件分支，调用 do_syscall 进行处理，返回参数放在 a0 中，由于这里只有一个用户程序，始终返回代表成功的 0，即对上下文 c 的 GPRx 成员赋值 0。层层返回完成处理后通过 mret 指令回到 cpu.mepc 处继续执行。

申请成功时 program_break 会根据 increment 发生相应的增减，然后根据成功与否决定调用 write 的参数。若成功则 count 为需要输出的字符数，失败则为 1。write 系统调用的过程与 brk 基本一致，write 的系统调用号为 2，参数分别为 fd（输出格式）、buf（输出位置）和 count（输出字符数），参数用 a0、a1 和 a2 保存，最后调用 do_syscall 处理，首先检查 fd 的值，若为 1 或者 2 则利用 putch 从堆区向 buf 处输出 count 个字符，并返回成功输出的字符数（因为输出过程中可能会被其他异常打断、或是发生输出区域不足的情况）。失败则返回-1。同样结束后返回系统调用的指令的位置继续执行。