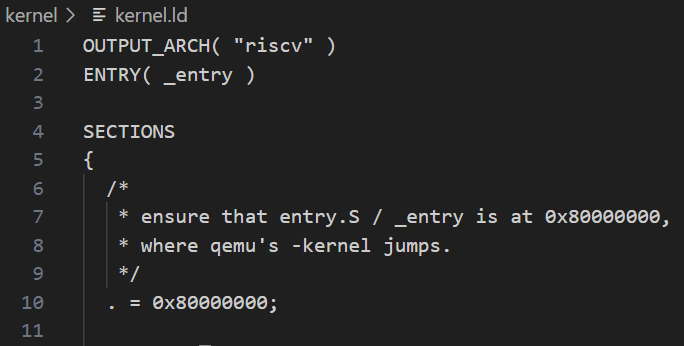
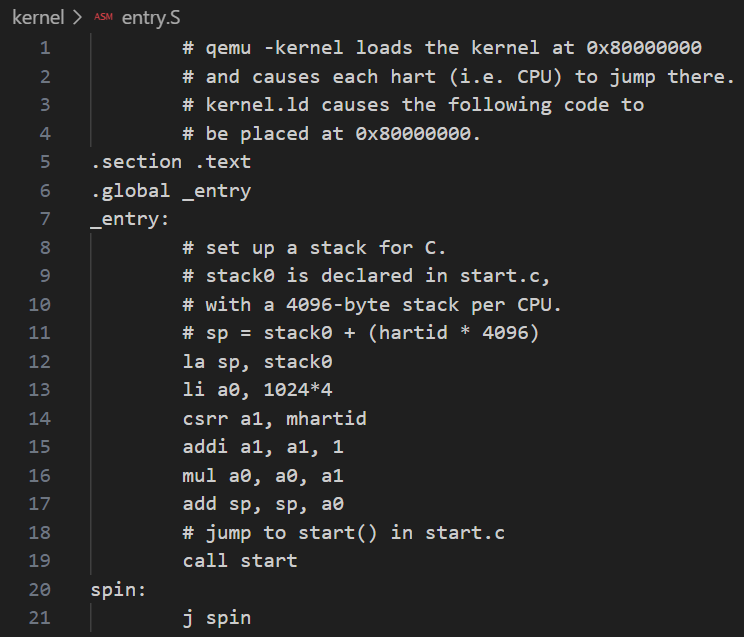
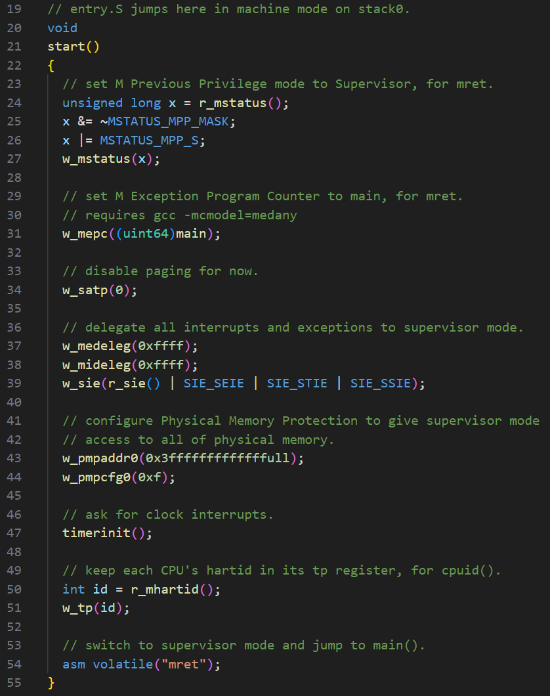
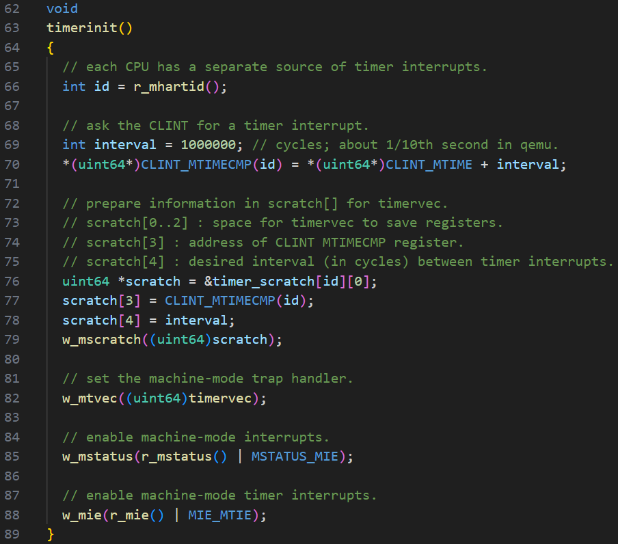
xv6操作系统启动过程

1. 操作系统开始运行之前，需要一段引导程序将操作系统kernel放入内存物理地址为0x80000000的地方，该地址在kernel.ld文件中设置。

上述部分代码的意思是告诉链接器生成适用于 RISC-V 架构的输出文件，并定义了程序的入口点是 \_entry 函数。在链接器脚本中，. 表示当前段的地址，= 0x80000000 表示将当前段（通常是代码段 .text）的起始地址设置为 0x80000000。这样，所有的代码和数据都将从该地址开始放置。

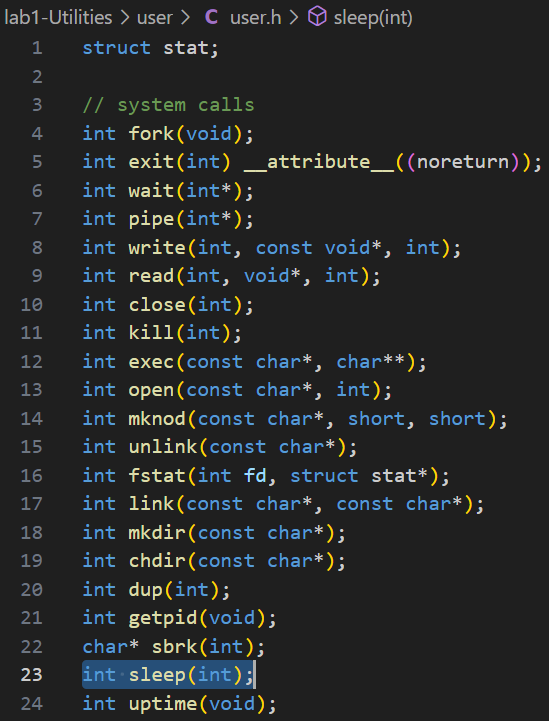
\_entry 函数代码如下图所示，其主要作用是按照CPUid (mhartid)，将栈指针寄存器(sp)赋值为stack0+hardid\*4\*1024，然后跳转到start()函数开始执行。

start()函数主要完成一系列寄存器的初始化，然后执行main()函数。为什么会执行main()函数呢？这部分具体由汇编指令mret实现，该指令会将寄存器mepc中的值加载到PC中，然后将控制权交给mepc处的代码。具体代码如下：

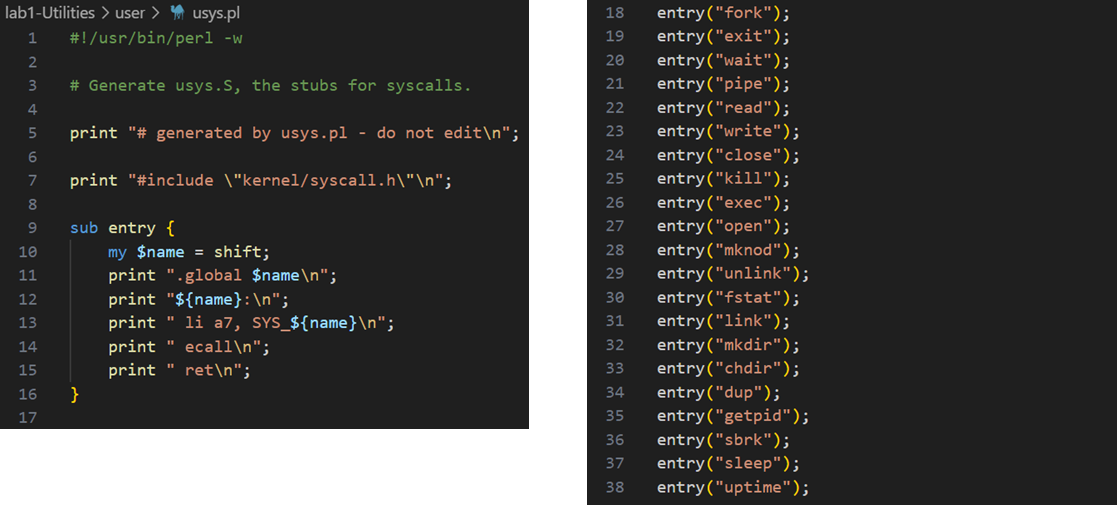


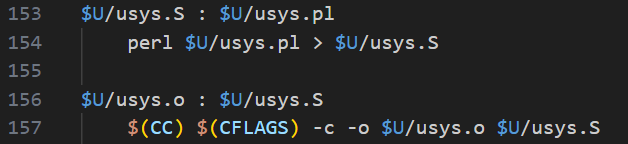
以Lab: Xv6 and Unix utilities中sleep为例解释**系统调用实现流程**

1. 用户态Sleep函数的实现

(1) sleep函数的声明

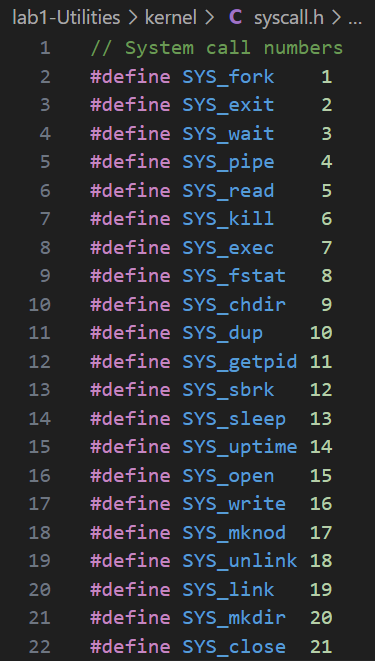
(2) sleep函数的实现

用户态Sleep函数实现较为复杂，下面一一道来。项目中有Perl文件user/usys.pl，文件内容如下：

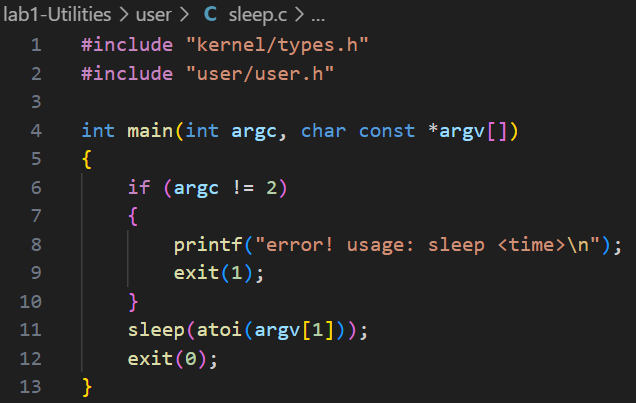
 Makefile中的命令：

我们可以看到，执行perl user/usys.pl并将输出结果重定向至user/usys.S文件中生成汇编语言，最终生成对应的可重定向文件。以entry(“sleep”)为例，汇编中.global sleep表明sleep是全局符号，可以在其他文件中调用。sleep:表明这是sleep函数的入口标签。接下来三句汇编是RISC-V指令，分别表明将SYS\_sleep加载到寄存器a7中，ecall执行系统调用，最后返回。

那么暂时我们还有一个疑问，SYS\_sleep是什么，又是在哪里定义的？我们可以看到kernel/syscall.h文件中有定义：

(3) sleep函数的调用

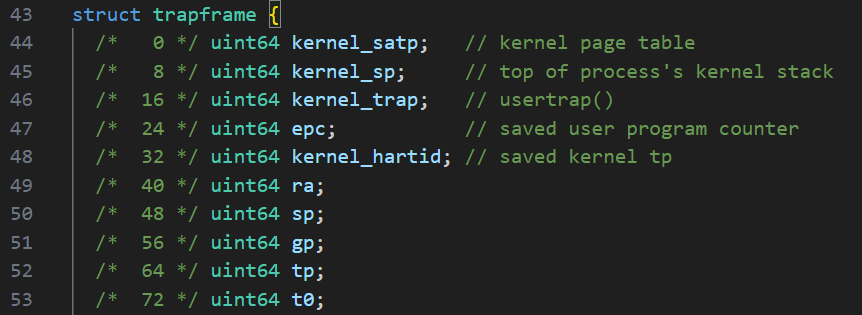
新建sleep.c文件，获取命令行输入，并调用sleep函数。

2. 系统调用指令ecall的实现

ecall是进行实际系统调用的入口。它是由RISC-V提供的一个用于实现系统调用的指令，通常由低特权的代码发起，用来执行高特权代码，比如UserMode到SupervisorMode、SupervisorMode到MachineMode。

下面(1)(2)部分，我们讲述一下预备知识。

(1) 首先要认识到，在执行汇编指令时，我们不需要在代码中定义寄存器，因为汇编指令转化为二进制之后可以直接被机器执行。例如汇编指令asm volatile("csrw sepc, %0" : : "r" (x));会通过硬件电路直接将相应的值写入寄存器sepc中。

 但是，为什么像kernel/proc.h中会定义一些包含寄存器的结构体呢？我们可以认为这部分寄存器的定义不是说RISC-V中就只有这些寄存器，而是表明在某些情况下这些寄存器的值需要被保存。

(2)

#define MAXVA (1L << (9 + 9 + 9 + 12 - 1)) // MAXVA = (1L << 38)，即0x8000000000

#define PGSIZE 4096 // 每页字节数，即0x1000

#define TRAMPOLINE (MAXVA - PGSIZE) // 起始地址：0x3ffffff000

#define TRAPFRAME (TRAMPOLINE - PGSIZE) // 起始地址：0x3ffffffe00

(3) RISC-V trap machinery 陷入机制

每个RISC-V CPU都有一组控制寄存器，内核写入这些寄存器来告诉CPU如何处理陷阱。实现寄存器的读取或写入的函数在riscv.h中定义，命名方式为“r/w”+“\_”+对应寄存器的名称，如w\_sepc(uint64 x)或r\_sepc()。下面介绍一些相关的寄存器，这些寄存器只能在supervisor mode时读写。

stvec : 内核在该寄存器写入trap处理程序的地址，RISC-V跳到stvec的地址0x3ffffff000完成相关操作。

sepc : 当trap发生时，RISC-V会在此处保存程序计数器（因为pc会被stvec中的值覆盖）。sret（return from trap）指令将sepc复制到pc。

scause : RISC-V在这里放了一个数字，描述了trap的原因。

sscratch :指向地址0x3ffffffe00，即trapframe开始地址，用来保存寄存器的值。

sstatus : SIE位控制是否启用设备中断。如果内核清除SIE，RISC-V将推迟设备中断，直到内核设置SIE。SPP位指示陷阱是来自user mode还是supervisor mode，并控制trap返回到什么模式。

当执行trap时，RISC-V硬件会对所有陷阱类型执行以下操作：

1如果是设备中断且sstatus的SIE位清零，则不做任何操作。

2通过清零sstatus的SIE位禁止中断。

3将pc的值复制到sepc中。

4在sstatus的SPP位保存当前状态是user mode还是supervisor mode。

5设置scause表明trap的原因。（8意思是）

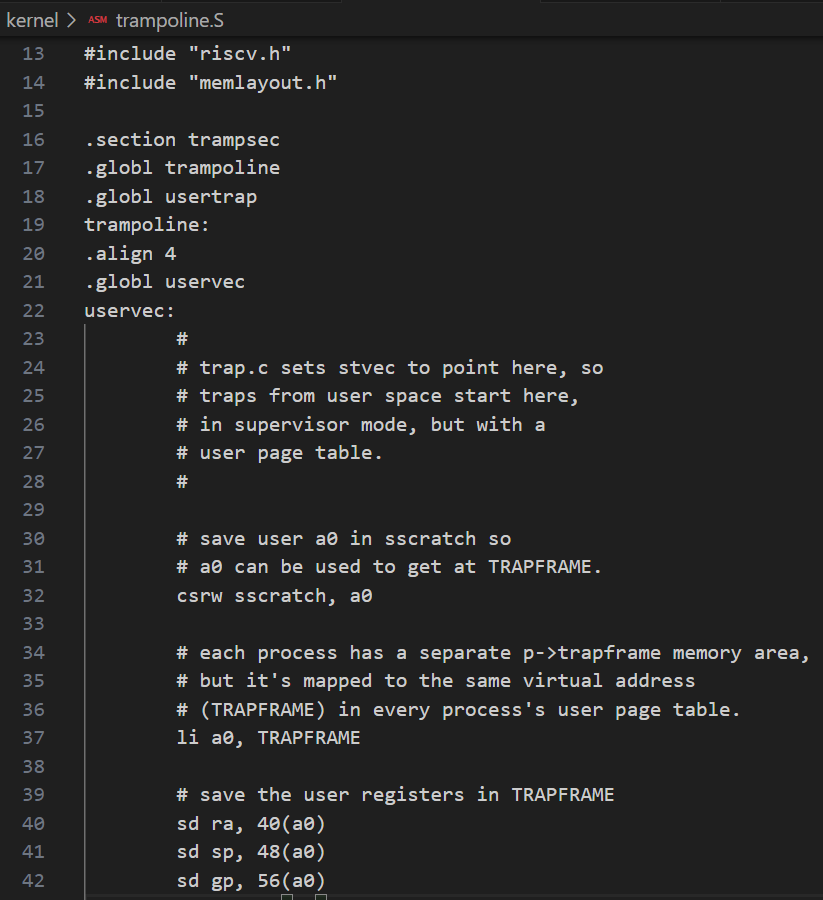
6设置supervisor mode。

7将stvec的值复制pc中。

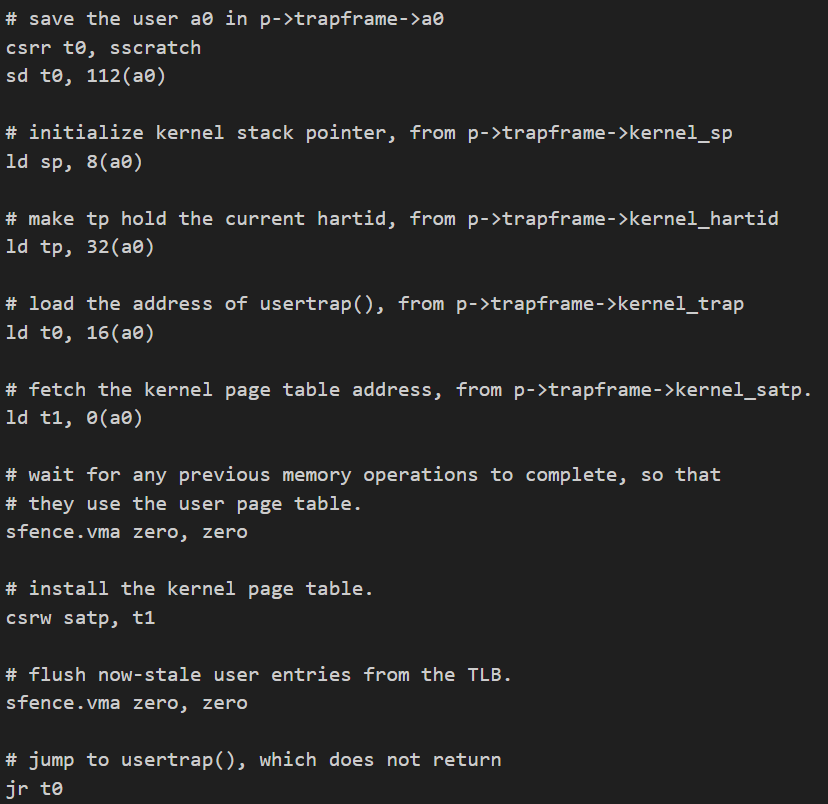
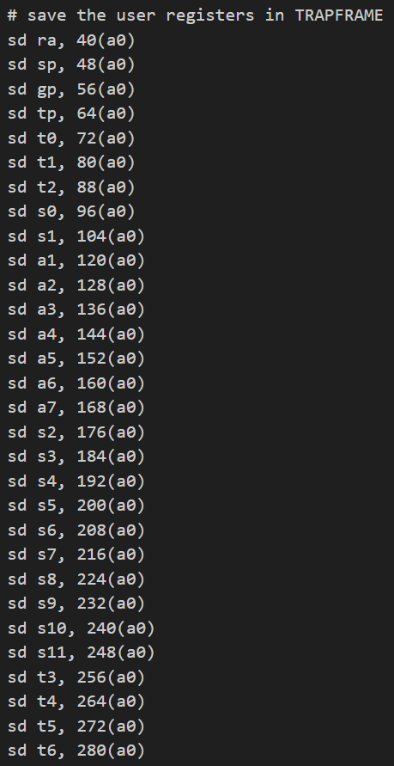
8从新的pc处开始执行

(4) 从用户态trap

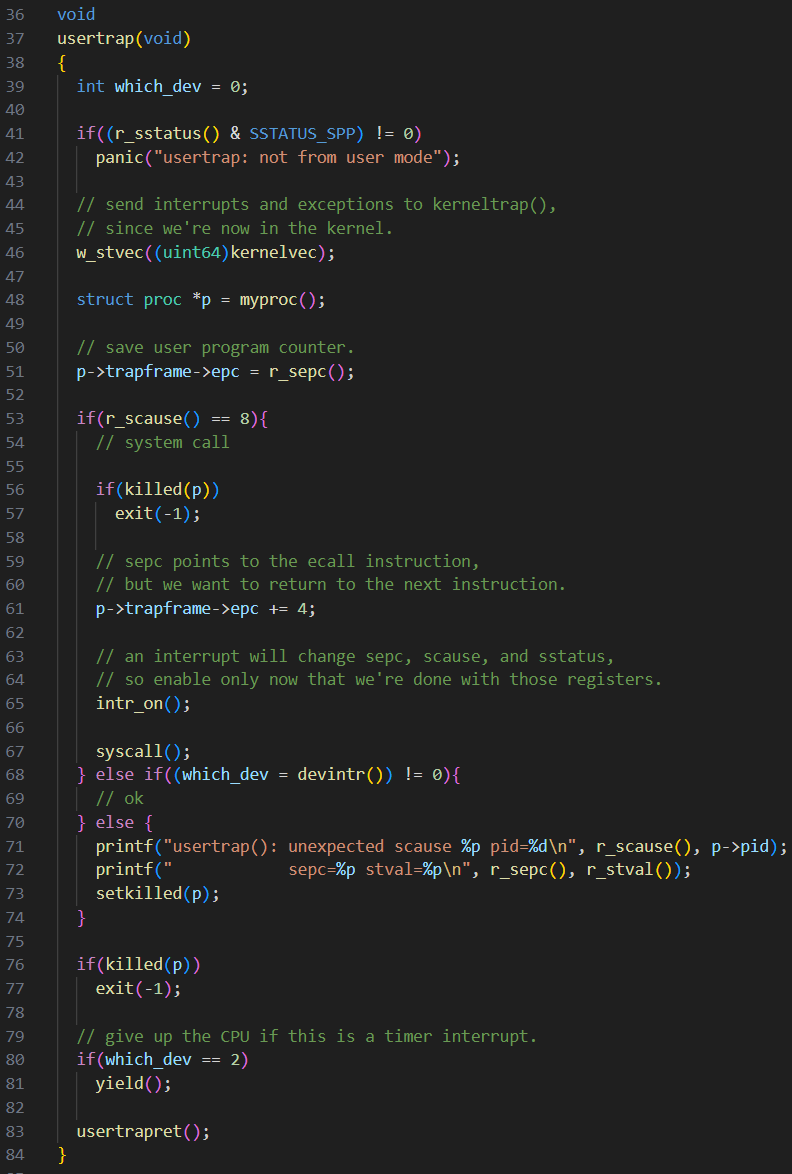
从上面可知调用ecall执行后程序会从stvec处开始执行命令，即地址0x3ffffff000处，即trampoline的初始位置。（在从内核空间进入到用户空间之前，内核会设置好stvec寄存器指向内核希望trap代码运行的位置。）

 地址0x3ffffff000的代码如下：

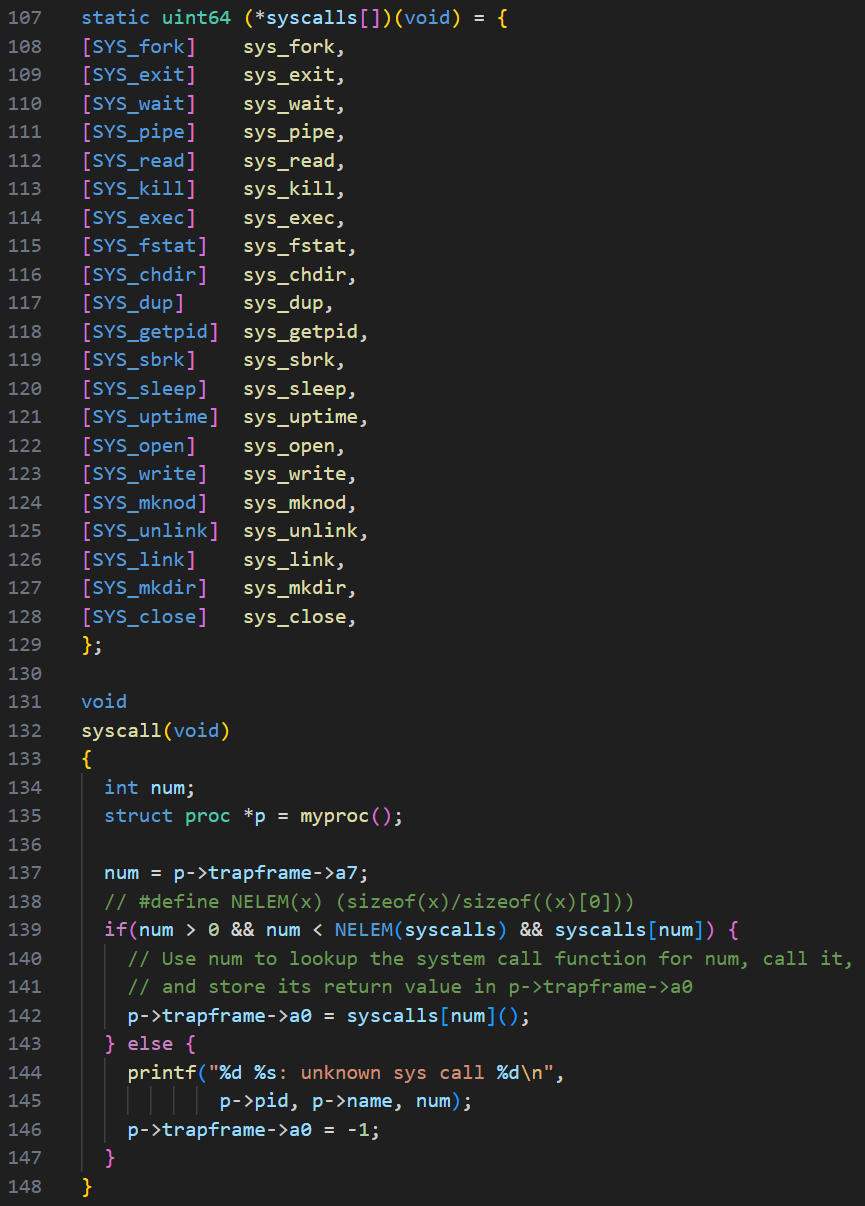
上述代码首先将a0的值写入sscratch中，然后将a0的值赋值为0x3fffffe000，即trapframe起始地址用于保存用户态寄存器的值。

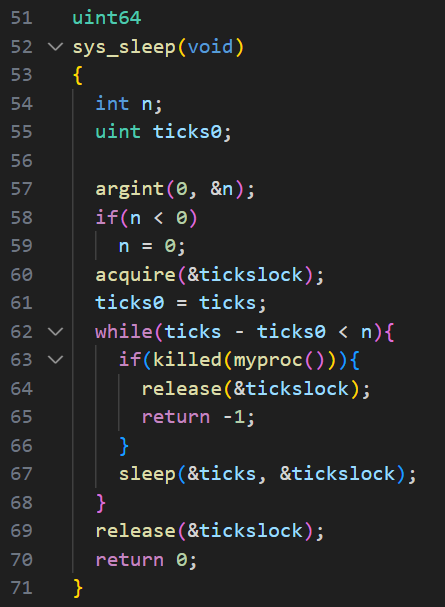


可以注意到左图我们没保存112(a0)的值，这是因为当前a0的值保存在寄存器sscratch中，直到右图前两句汇编语言才将a0的值保存到偏移量112的位置。

然后通过jr t0跳转到usertrap(void)函数：

usertrap函数中调用syscall()，syscall函数实现如下：

3. 内核态sleep函数的实现

syscalls[num]()调用sys\_sleep(void)函数完成系统调用。