3.7 编译运行kernel

接下来我会切换到代码介绍，来看一下XV6是如何工作的。

首先，我们来看一下代码结构，你们或许已经看过了。代码主要有三个部分组成：

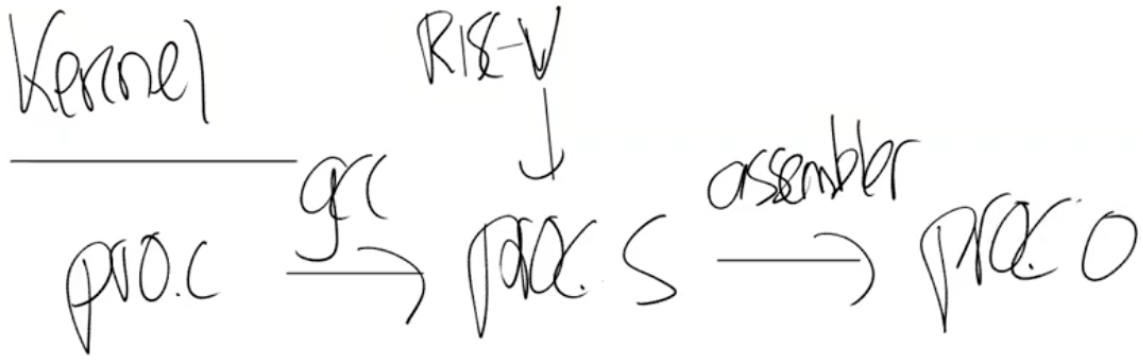
- 第一个是kernel。我们可以ls kernel的内容，里面包含了基本上所有的内核文件。因为XV6是一个宏内核结构，这里所有的文件会被编译成一个叫做kernel的二进制文件，然后这个二进制文件会被运行在kernle mode中。

```
[xv6-riscv (riscv)]$ ls kernel
bio.c       fs.c          param.h      spinlock.c   sysproc.c
bio.d       fs.d          pipe.c       spinlock.d   sysproc.d
bio.o       fs.h          pipe.d       spinlock.h   sysproc.o
buf.h       fs.o          pipe.o       spinlock.o   trampoline.o
console.c   kalloc.c      plic.c       start.c      trampoline.S
console.d   kalloc.d      plic.d       start.d      trap.c
console.o   kalloc.o      plic.o       start.o      trap.d
date.h      kernel        printf.c     stat.h       trap.o
defs.h      kernel.asm    printf.d     string.c     types.h
elf.h       kernel.ld     printf.o     string.d     uart.c
entry.o     kernel.sym    proc.c       string.o     uart.d
entry.S     kernelvec.o   proc.d       swtch.o      uart.o
exec.c      kernelvec.S   proc.h       swtch.S      virtio_disk.c
exec.d      log.c         proc.o       syscall.c    virtio_disk.d
exec.o      log.d         ramdisk.c    syscall.d    virtio_disk.o
fcntl.h     log.o         riscv.h      syscall.h    virtio.h
file.c      main.c        sleeplock.c  syscall.o    vm.c
file.d      main.d        sleeplock.d  sysfile.c    vm.d
file.h      main.o        sleeplock.h  sysfile.d    vm.o
file.o      memlayout.h   sleeplock.o  sysfile.o
[xv6-riscv (riscv)]$
```

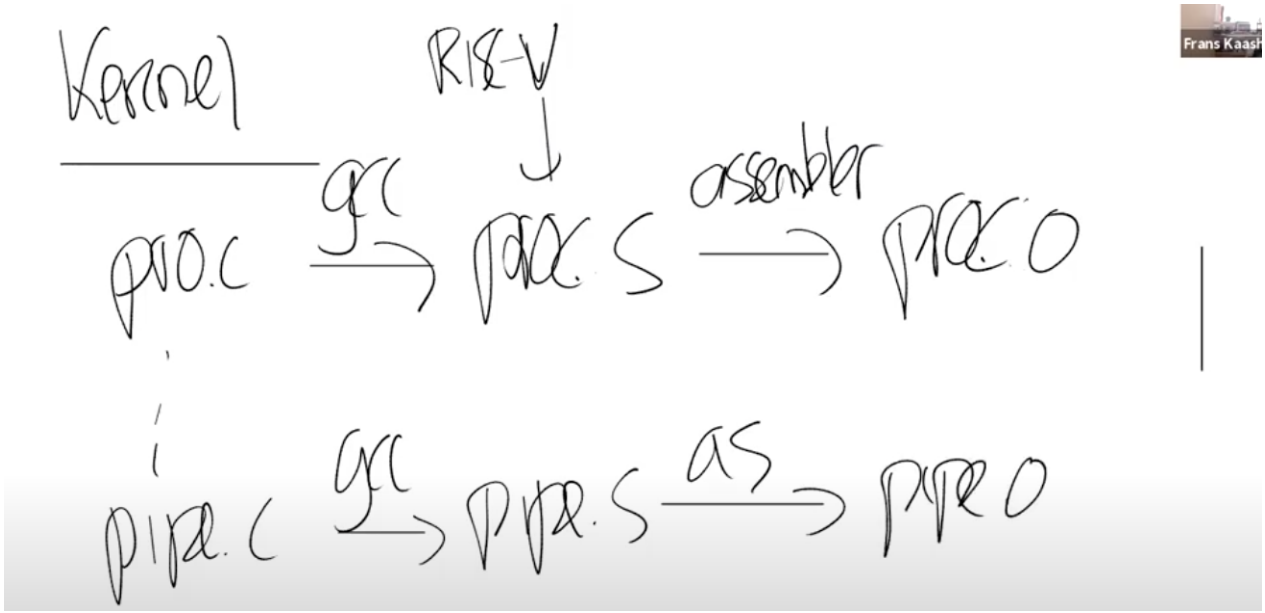
- 第二个部分是user。这基本上是运行在user mode的程序。这也是为什么一个目录称为kernel，另一个目录称为user的原因。
- 第三部分叫做mkfs。它会创建一个空的文件镜像，我们会将这个镜像存在磁盘上，这样我们就可以直接使用一个空的文件系统。

接下来，我想简单的介绍一下内核是如何编译的。你们可能已经编译过内核，但是还没有真正的理解编译过程，这个过程还是比较重要的。

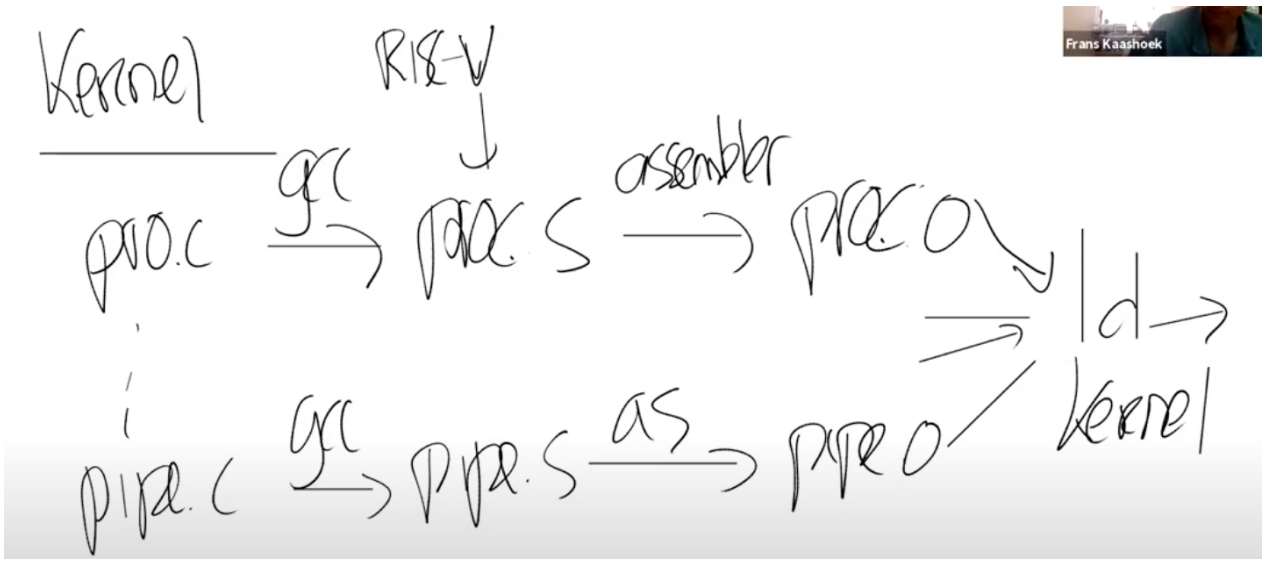
首先，Makefile（XV6目录下的文件）会读取一个C文件，例如proc.c；之后调用gcc编译器，生成一个文件叫做proc.s，这是RISC-V 汇编语言文件；之后再走到汇编解释器，生成proc.o，这是汇编语言的二进制格式。



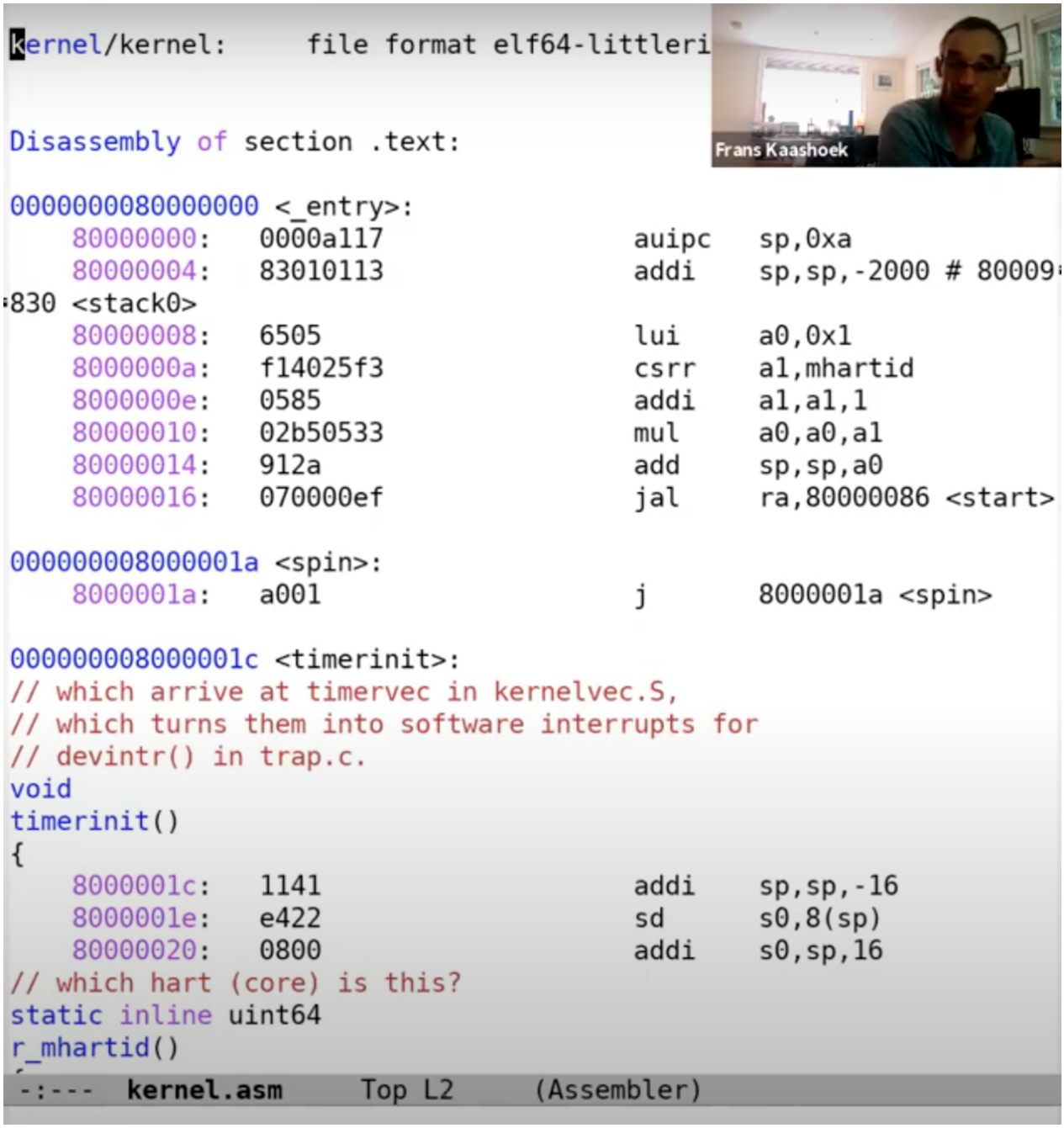
Makefile会为所有内核文件做相同的操作，比如说pipe.c，会按照同样的套路，先经过gcc编译成pipe.s，再通过汇编解释器生成pipe.o。



之后，系统加载器（Loader）会收集所有的.o文件，将它们链接在一起，并生成内核文件。



这里生成的内核文件就是我们将会在QEMU中运行的文件。同时，为了你们的方便，Makefile还会创建kernel.asm，这里包含了内核的完整汇编语言，你们可以通过查看它来定位究竟是哪个指令导致了Bug。比如，我接下来查看kernel.asm文件，我们可以看到用汇编指令描述的内核：

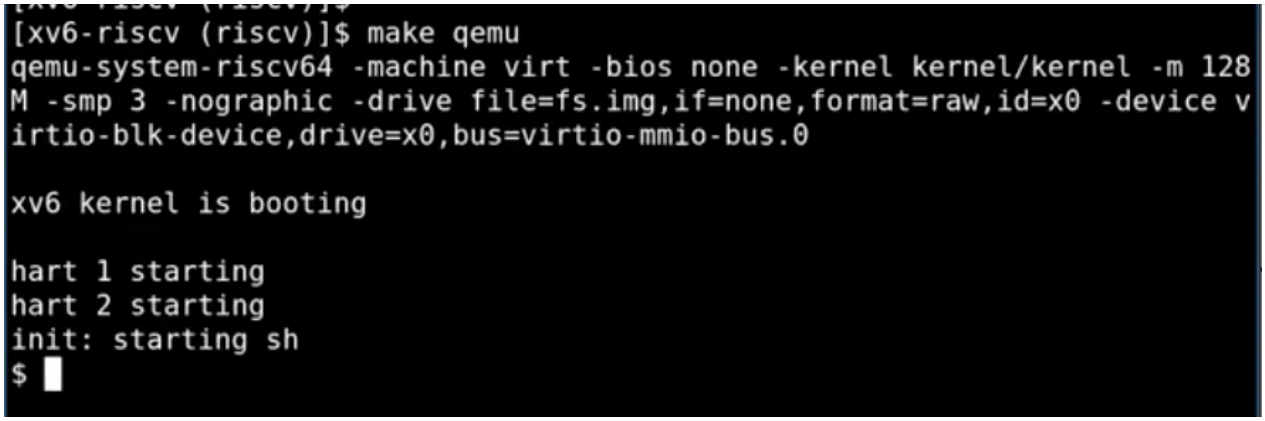


这里你们可能已经注意到了，第一个指令位于地址0x80000000，对应的是一个RISC-V指令：auipc指令。有人知道第二列，例如0x0000a117、0x83010113、0x6505，是什么意思吗？有人想来回答这个问题吗？

> 学生回答：这是汇编指令的16进制表现形式对吗？

是的，完全正确。所以这里0x0000a117就是auipc，这里是二进制编码后的指令。因为每个指令都有一个二进制编码，kernel的asm文件会显示这些二进制编码。当你在运行gdb时，如果你想知道具体在运行什么，你可以看具体的二进制编码是什么，有的时候这还挺方便的。

接下来，让我们不带gdb运行XV6（make会读取Makefile文件中的指令）。这里会编译文件，然后调用QEMU（qemu-system-riscv64指令）。这里本质上是通过C语言来模拟仿真RISC-V处理器。



我们来看传给QEMU的几个参数：

- -kernel：这里传递的是内核文件（kernel目录下的kernel文件），这是将在QEMU中运行的程序文件。
- -m：这里传递的是RISC-V虚拟机将会使用的内存数量
- -smp：这里传递的是虚拟机可以使用的CPU核数
- -drive：传递的是虚拟机使用的磁盘驱动，这里传入的是fs.img文件

这样，XV6系统就在QEMU中启动了。