XV6 源码阅读
中断异常机制报告

阅读代码

启动部分

Bootloader: bootasm.S bootmain.c: 系统加电启动的时候会首先运行存在 ROM 上的 BIOS，BIOS 会进行系统硬件的检查等，之后 BIOS 会通过 MBR（位于装有操作系统的磁盘上的主引导记录）及 DBR（分区引导记录）找到操作系统，此处的 bootsam.S 便是操作系统中最开始想运行在一个 CPU 上的部分，启动代码部分没有包括 efi 相关文件，据此可以说 xv6 仅支持 BIOS 启动方式。bootasm.S 由 16 位及 32 位汇编共同携程，它的作用主要是将 CPU 从实模式切换到保护模式，然后设置 GDT 后再调用 bootmain.c 中的 bootmain。boootmain.c 中主要定义了 bootmain 函数，它的作用是将操作系统内核的 ELF 文件的前 4096 字节加载到内存，然后通过这 4096 字节内容对 ELF 文件做判断后再将整个 ELF 文件加载进内存，然后调用其 entry。entry 定义在 entry.S 中，其主要作用是设置页及页目录后打开分页，再设置栈指针后调用 main.c 中的 main 函数。

XV6 初始化模块: main.c bootother.S: main.c 中的 main 函数包含了一系列的初始化动作（内存的初始化、中断控制器的初始化、进程表、文件管理表、磁盘的初始化以及缓存缓冲的初始化），并启动了系统中可能会有的其他的 CPU（每个 CPU 都会在初始化自己的 APIC 和 IDT 等后开始执行调度函数 scheduler）。

中断与系统调用部分

trap.c trapasm.S vectors.S & vectors.pl syscall.c sysproc.c proc.c 以及相关其他文件代码：

vectors.pl 用于生成 vectors.S，其中定义了中断 256 个中断描述符，每个中断描述符会将中断描述符号存入寄存器后跳转到 alltraps，而 alltraps 定义在 trapasm.S 中。

trapasm.S 包含过程 alltraps，其作用是创建一个陷入帧，在准备好数据和 CPU 寄存器后调用定义在 trap.c 中的 trap 函数。另一个过程

trapret 则是恢复寄存器后中断返回。

traps.h 定义了所有的陷入类型，64 为系统调用。

trap.c idt 中断描述符表（大小 256）类型为 gatedesc（定义在 mmu.h 中），即门描述符（可以使用 SETGATE 宏来设置）。

tvinit 函数和 idtinit 函数会在之前提到的启动时调用，tvinit 初始化中断向量，SETGATE 宏定义中 istrap（系统调用为 trap，其他都为中断），SEG_KCODE（Kernel 代码段），DPL（Descriptor Privilege Level）初始化自旋锁 tickslock。

trap 处理中断，中断转给 syscall 函数（定义在 syscall.c 中），然后再转给各中断处理函数（定义在 sysproc.c 中），做了对中断注册的实现。

spinlock.h 定义了自旋锁结构 spinlock，其中有记录持有锁的 cpu。

proc.c 中定义了有关进程的各种数据结构和处理函数，userinit 函数在启动初始化的时候被调用，用于启动第一个用户进程。allocproc 用于查找进程表中是否有空余的位置，分配系统栈及新的上下文信息。

syscall.c 其头文件 syscall.h 中定义了系统调用号，绑定了各系统调用到 syscalls 数组上。

sysproc.c 中实现了许多系统调用。

问题

1. 什么 是用户态和内核态？两者有何区别？什么是中断和系统调用？两者有何区别？计算机在运行时，是如何确定当前处于用户态还是内核态的？

用户态是一个进程运行自己的代码时的状态，是特权最低的级别，当进程执行系统调用的时候，系统会陷入内核态，这时候执行的并不是进程自己的代码，而是内核的代码，并且这时特权最高，例如在用户态不能直接进行对文件的操作，也不能进行网络数据的发送等操作，而是要通过系统调用完成。两者的主要区别便是特权等级不同，访问的堆栈也不同，这样有效地提高了系统的安全性。计算机在运行时是通过 x86 的特权级来确定处于内核态还是用户态的。

2. 计算机开始运行阶段就有中断吗？ XV6 的中断管理是如何初始化的？ XV6 是如何实现内 核态到用户态的转变的？ XV6 中的硬件中断是如何开关的？实际的计算机里，中断有哪几种？

计算开始运行阶段就有中断了，因为中断是 CPU 的硬件机制。中断管理的初始化：在 main.c 文件中的 main 函数中调用了 tvinit 来初始化中断描述符表 idt 并关联 vectors.S，然后在 main.c 中的 mpmain 函数中调用了 idtinit 设置了时钟中断。来 XV6 中实现内核态到用户态的转变：在 proc.c 文件中的 userinit 函数中进行了第一个进程的初始化，在这里设置了其中断帧，其中就包含了在 cs 和 ds 中设置了 DPL_USER 来设置用户态，之后该中断帧中保存的寄存器被换上 CPU 时便实现了从内核态到用户态的转变。XV6 通过 cli 来关闭硬件中断，通过 sti 来开启硬件中断。实际的计算机里，中断有三种，分别为：来自指令内部的中断，比如除零或者页缺失等。来自外部硬件的中断，由外部硬件向 CPU 发起的中断请求，如磁盘完成读取后返回时会向 CPU 发起中断。另一种中断是来自用户程序的中断，是用户程序通过访管指令来发起的中断，这种中断包含了系统调用。

3. 什么是中断描述符，中断描述符表？在 XV6 里是用什么数据结构表示的？

在 x86 中，中断描述符表中定义了中端处理程序的入口。在 XV6 里这个表有 256 个表项，每一个都提供了相应的 %cs 和 %eip。在 XV6 中用 idt 即为该表，它是结构体 gatedesc 的数组，门描述符 gatedesc 结构中即包含了内核代码段的段编号 cs 等。

4. 请以某一个中断（如除零，页错误等）为例，详细描述 XV6 一次中断的处理过程。包括：涉及哪些文件的代码？如何跳转？内核态，用户态如何变化？涉及哪些数据结构等等。

如果发生了除零，CPU 会检测到这个中断，并开始执行中断处理过程，首先会通过硬件完成中断描述符的获取、特权级检查和一些寄存器的压栈。然后根据 vectors.S 执行到 trapasm.S 中的 alltraps，alltraps 首先继续保存 CPU 寄存器中的上下文信息，然后设置数据和 CPU 的

段后调用 trap.c 中的 trap。trap 函数则是先判断该中断是否为系统调用，然后判断是否为硬件中断，由于除零不属于这两种并且发生在用户态，则默认执行将除零的用户进程杀死的操作。

5. 请以系统调用 setrlimit （该系统调用的作用是设置资源使用限制）为例，叙述如何在 XV6 中实现一个系统调用。（提示：需要添加系统调用号，系统调用函数，用户接口等等）。

系统调用号定义在 syscall.h 中，因此首先在 syscall.h 中添加 setrlimit 的定义#define SYS_setrlimit 22。然后在 syscall.c 中添加系统调用的处理函数的调用，即在[SYS_setrlimit] SYS_setrlimit。在 sysproc.c 中添加系统调用函数 int sys_setrlimit(void)来具体实现系统调用的功能。最后在 user.h 中声明该系统调用的接口 int setrlimit()。在 usys.S 添加 SYSCALL(setrlimit)，这样在程序中便可以使用 setrlimit 系统调用了。