Homework 2: understand **xv6**'s interrupt and exception handling procedures

中断子系统是操作系统中的重要一员。当在用户态中发生中断、异常、系统调用都会导致系统强制暂停当前任务、进入内核态，执行对应的服务例程，并作出相应的处理。比如在用户态发生除零异常后，内核一个通常的做法是向这个用户态程序发送一个信号。

本次作业的目标是通过阅读 xv6 x86 版本的代码来理解系统在用户态发生 trap（中断、异常、系统调用等导致）之后是如何处理的。为了更好地理解 xv6 的处理过程，我们假设当前 xv6 的用户程序发生了除 0 异常，分析在该场景下 xv6 会如何处理。

首先，用户程序发生除 0 异常，由于涉及到特权级的转换，会从用户栈切换到内核栈，CPU 硬件会：

1. 关闭中断；
2. 临时保存 SS（栈指针）、ESP（栈顶指针）、EFLAGS（标志寄存器）、CS（代码段指针）、EIP（程序计数器）寄存器的值；
3. 切换到内核栈；
4. 将 SS、ESP、EFLAGS、CS、EIP 寄存器的值压入到内核栈顶；
5. 如果发生了错误，还会向内核栈顶压入 error code。根据 x86 开发者手册 5-28 内容，可以发现除 0 异常并没有 error code；
6. 找到除 0 异常的 ISR（vectors.S 文件中的 vector0 函数），开始执行；

在 vector0 函数中，xv6 首先向内核栈手动压入 error code，然后压入对应的 trap number（标识 trap 的发生原因），最后调用到 alltraps 函数（trapasm.S 文件）。

本次作业的内容是分析后续的执行流程，简要说明 xv6 做了哪些操作、是如何处理发生除 0 异常的用户程序的。

xv6 中有一些汇编文件是在编译过程中生成的，作业压缩包中提供了编译完成的完整代码，并且为了更方便同学们理解源代码，添加了一些注释。你只需要阅读 trapasm.S 文件中的 alltraps 函数、trap.c 文件中的 trap 函数、proc.c 文件中的 exit 函数即可。

提交内容

一份 pdf 文件。

供参考内容

[1] CPU registers in x86: https://wiki.osdev.org/CPU_Registers_x86

[2] x86 and amd64 instruction reference: https://www.felixcloutier.com/x86/