Name: Graeme Hosford
Student ID: R00147327

XV6 System Calls and Interrupts Summary

In XV6 there are 3 cases when control must transfer from a user program to the OS kernel. These are when a system call is made, when an exception occurs and when an interrupt occurs when the device signals it needs attention from the kernel.

The kernel handles all interrupts as it is usually only the kernel which has the required privileges and state, for example this would include managing time slicing on which processes get CPU access. Some processes may continue to use the CPU without giving time to others and so the kernel must be involved to manage this.

When an interrupt occurs the normal processor loop is halted and a new sequence called the interrupt handler starts executing. Before this execution begins the processor saves the contents of its registers so the OS can restore them when it returns from its interrupt.

In XV6 these interrupts (or traps as they are also known in XV6) usually occur when a process makes a system call. There must then be a switch from user mode to kernel mode in order to execute this system call as user mode typically would not have the required privileges to do so.

In x86 there are 4 levels of privileged access numbered 0 (most privilege) - 3 (least privilege), in practice only these two levels are used as 0 (kernel mode) and 3 (user mode) with levels 1 and 2 being ignored. The current privilege level is stored in the %cs register in the CPL field. x86 also defines its interrupt handlers in the Interruptor Descriptor Table (IDT). This table holds 256 entries each one giving a %cs and %eip register to use when handling a corresponding interrupt.

When a user process on x86 makes a system call the program calls the int n instruction where n corresponds to the index in the IDT. After the process of retrieving the values from the IDT the %eip register is then pointing at the address specified in the IDT and this is the next instruction to be executed.

When a system call includes arguments these are not copied from the user stack onto the kernel stack, instead helper functions such as argint, argstr and argptr read the %esp register and locate the argument from there to be used in the system call.