Assignment Two Theory Questions

Gearóid Sheehan R00151523

Operating Systems Engineering COMP8051

1) – Read chapter 6 of the xv6 book and briefly explain how the read calls in the following code taken from cat.c are associated with sectors on the disk by the xv6 operating system.

When the cat command is entered into the terminal in xv6, the main loop in cat.c checks the number of arguments entered with the command and calls the cat function with the appropriate file descriptor for the file passed as arguments, if any. A while loop iterates, calling the read function retrieving 512 bytes at a time until it returns size zero bytes, meaning all the data has been retrieved. The purpose of the read() function is to read and count up the 'n' bytes from the system level into the user level buffer 'buf' from the given file descriptor 'fd' in cat.c.

In the system level, the syscall() function calls another function called sys_read(). This is the system level read in the OS. A struct called 'proc' is created for the cat.c process in the pTable, and the file descriptor for the given argument is passed into the 'struct file *ofile[NOFILE]' pointer as an index, which points in turn to a struct file. This struct file contains a pointer to an 'inode' data structure for the given file.

The function readi() reads in sectors from the buffer layer and into the given inode. There is a buffer at the block layer which has two jobs – to cache popular blocks and to synchronize disk access using a least recently used eviction policy (LRU). When xv6 attempts to retrieve a files data in a given inode, the buffer is checked to see it contains any of the blocks listed in the inodes 'addrs' array. If an address exists in the buffer, then the data for that block is also in the buffer and can be retrieved from there. If it does not exist, the LRU policy comes into play, removing the least recently used block from the buffer, creating a blank block which is flagged as dirty. This blank block is populated using the bread() function which calls the interface connecting the buffer caches with the disk layer, called iderw(). When the block is populated with the data it is flagged as valid as it now corresponds to a sector on the disk, as a copy in memory.

2 a) - Explain how the xv6 operating system uses interrupts to schedule I/O requests to the disk?

I/O requests made in the xv6 operating system work very similarly to system calls. The main difference is that interrupts from I/O devices can occur at any time, for example when a mouse is clicked or a button on a keyboard is pressed. The kernel handles all interrupts as it has the required privilege and state. Interrupts are used as opposed to polling, as polling involves a protocol where the CPU steadily checks the device for new events, which wastes CPU time when the events are infrequent.

Devices are programmed to generate an interrupt when an event occurs, which the motherboard receives and in turn informs the CPU. Xv6 is designed for a board with multiple processers. Each device is managed in the operating system by a driver which tells the

hardware to perform operations, as well as configuring and handling interrupts. The disk driver copies data in from the disk in 512 bytes into the OS to the struct buf, ensuring the operating system remains consistent in between reads as the structure will become out of sync with the disk. The driver can handle block sizes (BSIZE) that is a multiple of the sector size, although xv6 chooses it to be identical to the IDE's sector size. There are flags which track whether the data needs to be read in or written out using the B_VALID and B_DIRTY tags respectively. When the kernel is booted it calls ideinit in main.c which in turn calls ioapicenable to enable the IDE_IRQ interrupt. The disk is then checked by ideinit, calling idewait to wait for the disk to accept commands. Now that the disk controller is ready, ideinit can check how many disks are present. It assumes that disk 0 is present, because the boot loader and the kernel were both loaded from disk 0, but it must check for disk 1. It writes to I/O port 0x1f6 to select disk 1 and then waits a while for the status bit to show that the disk is ready. If not, ideinit assumes the disk is absent. When ideinit has run, the disk is not used again until the buffer cache calls iderw, which updates a locked buffer as indicated by the flags. If B_DIRTY is set, iderw writes the buffer to the disk; if B_VALID is not set, iderw reads the buffer from the disk.

2 b) - Explain how the bootloader interfaces with the IDE controller to load the xv6 operating system – see Appendix B of the xv6 book (section "Code: C bootstrap" in particular) and bootmain.c in the xv6 source code?

The bootloader is located in the first 521 bytes block of the disk, known as the bootsector, and its responsible for booting into the operating system when the computer is powered on. The bootloader interfaces with the IDE controller to load the xv6 operating system by first running its main function bootmain.c. The kernel executable is found in the struct elfhdr *elf.

The ELF headers are accessed when bootmain loads the first 4096 bytes of the ELF binary into memory. Its type is confirmed and bootmain begins to read the sections content in the disk location after the header's location. The memory is written to the address paddr. Bootmain calls readseg to load data from disk and calls stosb to zero the remainder of the segment. Stosb uses the x86 instruction rep stosb to initialize every byte of a block of memory.