Interactions Between Devices and the Operating System

CSCI460 - Montana State - Fall 2020

Chris Tompkins : z51s285

Samuel Forbes : h16n294

Mason Dinardi : w58p213

Brady Cornett : g46k828

Table of Contents

Abstract:

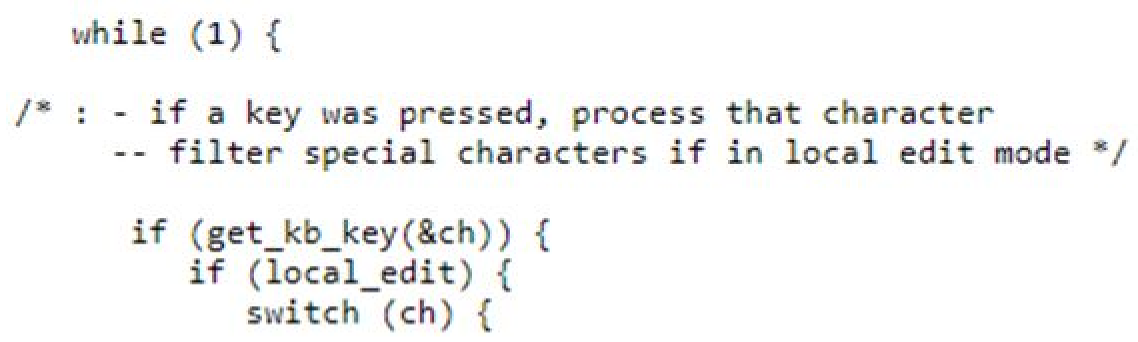
The purpose of this report is to provide insight into the process an operating system follows when utilizing several different forms of devices. Topics covered within this report include the history of device integration and research into how network, tertiary and storage devices cooperate with an operating system. The report contains screenshots of source code and explanations for each section. There is a heavy focus on drivers due to their critical role they play in the interaction between devices and an operating system.

History of Device Integration:

Early Device Interaction

The earliest computers were mainframe computers that had no real operating systems. Input/output devices consisted of punch cards and magnetic tapes. Punch cards were literally just pieces of stiff paper that contained data represented as either a hole or the absence of. This form of input and output proved incredibly annoying and error prone, thus the magnetic tape was born. In 1951 the Univac File-Computer utilized Mylar magnetic tape as its medium for input/output. This technology was a huge improvement in the world of computing, as the computer could share its operating system with the storage drums and with the central computer. This meant that a program would be available in the memory of the computer at all times.

In 1964, the first computer system that included a keyboard was introduced by MIT and Bell Laboratories. This was called Multics and allowed a user to see the text they were typing on a display screen which they could then edit. Multics used an I/O Daemon to control all printers and card equipment in the background. MIT released the source code for the final Multics release of 1992 so we could not get the original source code for the keyboard handler, but we did find the updated version. This was actually updated to C code throughout its lifetime so, thankfully, it was able to be interpreted. In a nutshell, each key has a unique electrical signal. When a key is pressed, the electrical signal is picked up by the OS through a file called getkbkey, which will hold the keypress as an ASCII value in a keyboard buffer. The next call is to an interpreter called ascii_to_abcdef_, which maps the ASCII value in the buffer to a certain letter or command. This repeats unconditionally as you can see in the snippet below:



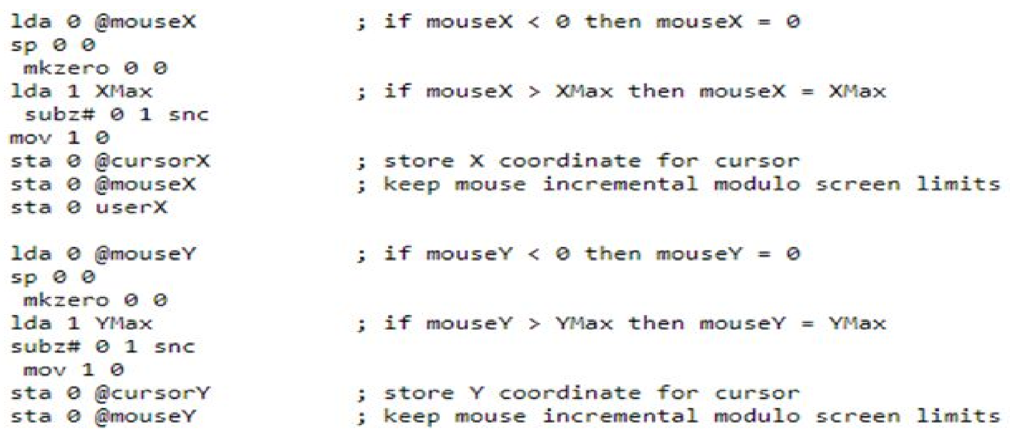
```
   while (1) {

/* : - if a key was pressed, process that character
     -- filter special characters if in local edit mode */

      if (get_kb_key(&ch)) {
         if (local_edit) {
            switch (ch) {
```

After keyboards, the next piece of hardware born was the mouse. The creation of the mouse is credited to Douglas Engelbart of the Stanford Research Institute. Engelbart wanted to create something more natural and easier to use than a stylus and thus the mouse was born. The first computer that was sold with a mouse was the Xerox Alto. The Xerox Alto was one of the first computers to put together a total Graphical User Interface using mouse integration.

The software to run the mouse is all found within the operating systems keyboard handler. The Alto Operating System Reference Manual states that "the keyboard handler periodically copies the mouse coordinates into the cursor coordinates", basically the mouse cursor will send the OS its X and Y coordinates on the screen boundary, and the OS will then store the coordinate in memory to recognize the location of clicks on screen. (Xerox, 1980, p. 8) This can be shown here in this code snippet written in ASM:



```
lda 0 @mouseX           ; if mouseX < 0 then mouseX = 0
sp 0 0
 mkzero 0 0
lda 1 XMax              ; if mouseX > XMax then mouseX = XMax
 subz# 0 1 snc
mov 1 0
sta 0 @cursorX          ; store X coordinate for cursor
sta 0 @mouseX           ; keep mouse incremental modulo screen limits
sta 0 userX

lda 0 @mouseY           ; if mouseY < 0 then mouseY = 0
sp 0 0
 mkzero 0 0
lda 1 YMax              ; if mouseY > YMax then mouseY = YMax
subz# 0 1 snc
 mov 1 0
sta 0 @cursorY          ; store Y coordinate for cursor
sta 0 @mouseY           ; keep mouse incremental modulo screen limits
```

Driver Overview

Hardware devices like the keyboard and mouse are controlled through a software interface called device drivers. Device drivers communicate with a hardware feature through the computer bus or a subsystem that the hardware connects to. A routine is invoked in the driver and the driver issues commands back and forth sending data. Basically, a driver is a bridge between hardware and software.

The operating system typically comes with the software for a Logical Device Driver, while the actual device will have a physical device driver. A logical device driver is an API that typically assumes how a device will behave and talks to the physical device driver. The API will then decide whether or not the request from the physical device driver is valid or not.

The early keyboards and mice mentioned above did not use logical device drivers or physical device drivers. They primarily used real mode drivers to do the work, which are loaded in at the Disk Operating System (DOS) level or in the case of the Multics and Alto, are already installed in the machine. The real mode driver memory addresses actually correspond to real locations in memory. This means that since there is no virtual memory space to write to, these computers could not support multitasking or different privilege levels of code. Also, hardware is controlled directly without any safeguards, so there was a lot of hardware malfunctioning back then.

Shortly after real mode drivers, protected mode drivers came in to play. Protected mode means that direct access to devices is restricted to the operating system and certain privileged programs. This means that if someone were to hypothetically write a program to access an I/O port, the system would halt. However, in Linux if you use the root account or your program has super-user privileges, you are able to access these ports. Protected mode is the standard for most modern operating systems, for example, everything after Windows 98, and all versions of Linux employ this operation. In the next sections, we will be talking about how different network, storage, and tertiary devices interact with the operating systems through device drivers.

Network Devices:

Network devices are devices that connect several different computers together. There are many different types of network devices that a computer uses. These can be external to the computer, such as routers, modems or switches. Other types of devices include ethernet cards/NICs(network interface card) or WiFi cards. All of these different devices have a job in connecting a computer into a network.

Network devices interact with the operating system on a host machine by utilizing network device drivers. The drivers used to power the interaction between network devices and the operating system are kernel level drivers. Our research led us to two implementations of kernel level network drivers. Windows implements NDIS (Network Driver Interface Specification) while Linux uses the ifnet interface from BSD Linux. There is a project called “NDISWrapper” to provide a module around NDIS to allow the Linux kernel to utilize NDIS.

We focused heavily on the Linux system in regards to the relationship between the kernel and network drivers. During packet transmission, the linux kernel will utilize a network driver method to add each individual packet onto an outgoing queue. The driver is then in charge of sending the packet through the hardware (usually an NIC), to the network the computer is attached to.

During packet reception, network drivers use either an interrupt driven or polled process to notify the OS that a packet was received. The interrupt driver technique is most common.

Network drivers have to be prepared for failure and delay during many different segments of this process. There could be an issue with the kernel/OS sending packets to the driver itself, which needs to be handled and reported in a manner that is easy to identify. There is also the fact that hardware fails. A common solution to monitoring network driver interactions is using set timeouts when certain processes start. When these timeouts happen, the driver has to reset everything to make sure it is ready for further transmissions.

Storage Devices:

Storage devices are a necessary part of the fabric of an operating system. After all, without a storage device the operating system wouldn't have a place to store its executables. The two primary specifications that a computer abides by to exchange information to and from the hard drive are ATA or SCSI (Hard disk drive interface, 2020). Communication to DVD-ROMs or CD-ROMs can be exchanged in the same way except that ATA is replaced by an extension of itself called ATAPI (Contributors to Wikimedia projects, 2012b). Flash drives don't quite follow the same protocols. Communication via the USB to the flash drive can be found in the Universal Serial Bus specifications (Garney & Lueker, 2000).

As far as hard drives and optical disc drives are concerned, the type of standard or specification that your computer needs to use depends on the type of hard drive or disc drive you purchased or already have installed in your computer (Contributors to Wikimedia projects, 2012a). On Windows, this information can be found via a series of Win32 commands (Cossu, 2011). On Linux, it can be simply found via the smartctl command (Smith, 2014).

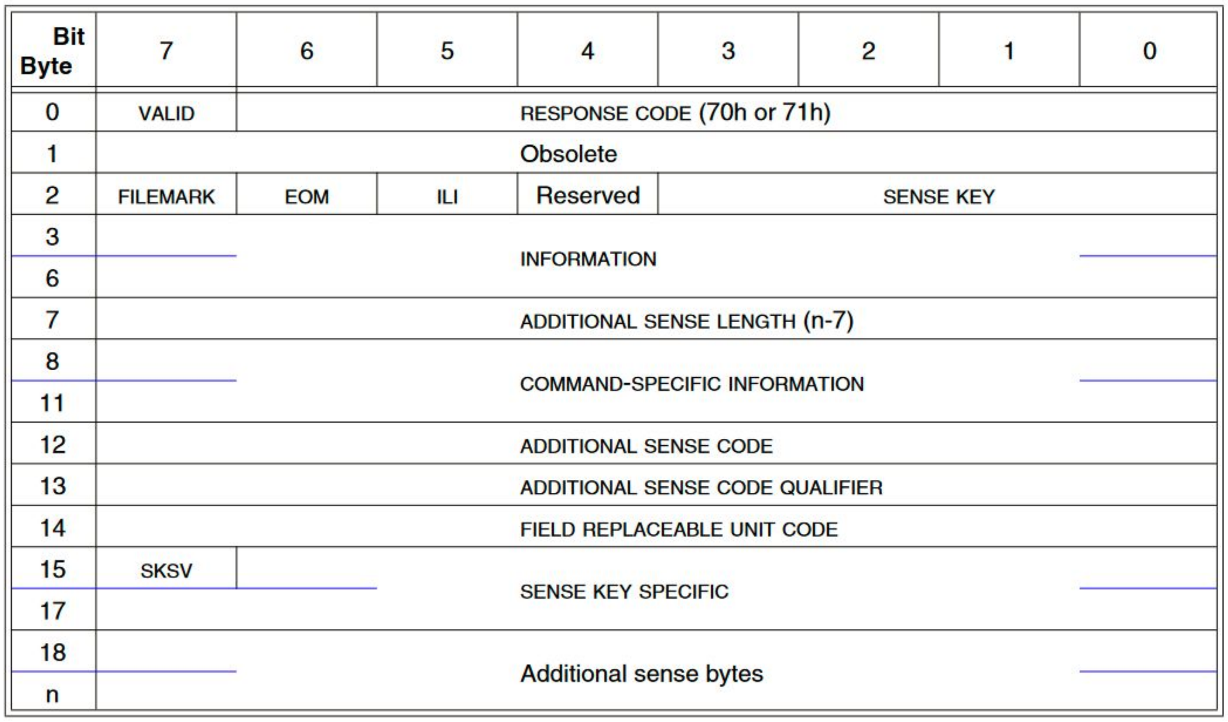
ATA is short for AT Attachment (Contributors to Wikimedia projects, 2020a). In SCSI, Small Computer System Interface, the operating system interacts with the storage device by sending 6-byte, 12-byte, or 16-byte packets to the operating system (Weber, 2008). These types of commands consist of reading data from the device, writing data to the device, sending diagnostic data to the device, receiving diagnostic data from the device, and changing mode pages, among many other things (Contributors to Wikimedia projects, 2020b).

Typically, the SCSI device has the ability to return data to the application by a few different types of methods, which were discussed in class. The operating system should be able to communicate with the SCSI device in all three methods we learned in class. However, I only seemed to come across two types of methods. One of the methods is called direct memory access, or DMA. In this mode, the device bypasses the processor and directly writes data to memory. The other “scheme” I came across was Programmed I/O, in which the processor polls the I/O device to see if more data can be retrieved (Peters, 2020; lorihollasch, 2018; see margin of MSDN doc for the other method).

When the application or operating system asks the storage device for information, it does so by sending a “command block” to the device. As was said before, for SCSI, this command block is either 6 bytes, 12 bytes, or 16 bytes in length. The first byte of the block consists of the opcode for the type of command to be used (Weber, 2008). This opcode indicates whether a read command should be used, or a diagnostic command, or some other command (Weber, 2008; Contributors to Wikimedia projects, 2020). The remaining bytes are specific to the command being called (Contributors to Wikimedia projects, 2020).

If the command fails, sense data is returned to the application or operating system. The verbose fixed part of fixed sense information has a size of 18 bytes (Weber, 2008, p. 226). In Windows, the application will have returned to it the sense information some time after the command has been fully processed by the storage device (lorihollasch, 2018). At a low level, however, a separate command is used to send the error information. That information is returned by the REQUEST SENSE command (Weber, 2008, p. 226). The type of error information that your drive is experiencing can be typically found out by the Additional Sense Code (ASC) and Additional Sense Code Qualifier (ASCQ), bytes 12 and 13 (zero-based) of the previously mentioned 18 bytes (Weber, 2008, p. 61).

Table 26 — Fixed format sense data



Bit / Byte	7	6	5	4	3	2	1	0
0	VALID	RESPONSE CODE (70h or 71h)						
1	Obsolete							
2	FILEMARK	EOM	ILI	Reserved	SENSE KEY			
3 – 6	INFORMATION							
7	ADDITIONAL SENSE LENGTH (n-7)							
8 – 11	COMMAND-SPECIFIC INFORMATION							
12	ADDITIONAL SENSE CODE							
13	ADDITIONAL SENSE CODE QUALIFIER							
14	FIELD REPLACEABLE UNIT CODE							
15	SKSV	SENSE KEY SPECIFIC						
16 – 17								
18 – n	Additional sense bytes							

(Weber, 2008, p. 61)

Some SCSI commands allow you to send mode pages to the device. A mode page is particularly powerful because it can be used to change how a storage device operates (Weber, 2008).

In Windows, SCSI commands can be sent via two IOCTLs (input/output controls): IOCTL_SCSI_PASS_THROUGH and IOCTL_SCSI_PASS_THROUGH_DIRECT (lorihollasch, 2018a). IOCTL_SCSI_PASS_THROUGH_DIRECT is used to have the device write the data to be retrieved directly to memory, bypassing the processor. IOCTL_SCSI_PASS_THROUGH writes the data with Programmed I/O (lorihollasch, 2018a).

For ATA, a similar process is used. However, the commands are slightly different. The first block of data indicates the feature that the command is included under. The next (typically 1-2 bytes) indicate a count (for example you may have to specify the number of times you want a command to occur). The next (typically) 3.5 bytes deal with the LBA that is being referred to. The LBA stands for the Logical Block Address (Contributors to Wikimedia projects, 2020c). The first LBA is 0 and each increment of 1 to it indicates that a new block is to be addressed. You can multiply the LBA by the block size to get the offset in bytes that is being referred to by the command. For hard drives, which ATA commonly addresses, a common block size is 512 bytes (Contributors to Wikimedia projects, 2020c). The next byte indicates certain flags that the device holds (Weber, 2013). Finally, the last byte is the command that is being used. Whereas SCSI used the first byte for the command, ATA uses the last byte to specify the command (Weber, 2013). As with SCSI, in Windows, there are two commands that can be used to send ATA commands to the device .

The two commands are IOCTL_ATA_PASS_THROUGH_DIRECT and IOCTL_ATA_PASS_THROUGH (lorihollasch, 2018b). ATAPI, AT Attachment Packet Interface, being an extension of ATA, uses those same commands to communicate to DVD-ROMs and CD-ROMs.

Tertiary Devices:

What is a tertiary device? We decided to use this section to cover a wide umbrella of devices that an operating system could interact with. This includes keyboards, mice, printers, bluetooth devices, webcams, and the like. These devices make interfacing incredibly straightforward and convenient for the user, almost by magic, but after taking this class, we know better. Operating systems are able to use pre existing drivers to seamlessly integrate most system hardware and simple devices. More complex devices, such as a fancy new light up keyboard or high end printer, might need a third party driver to be installed from the company's site. Most operating systems will recognize a PCI or USB device through two ID numbers consisting of 4 hexadecimal numbers each (Huang, A., & Rudolph, L. (2005). Bluetooth for Programmers). A vendor ID that identifies the vendor of the product, and a device ID that helps identify the specific product from said vendor.

This allows the operating system to integrate the device with seemingly no effort. In reality, the device driver interacts transparently with a hardware device, and usually provides the requisite interrupt handling necessary for any time-sensitive, asynchronous, hardware interfacing (DriversExpert, 2017).

Devices, most commonly printers, can have drivers embedded into their firmware. These are available to the operating system through a networking protocol. This method eliminates the need to install a driver physically onto the computer by having the computer accept the print data in a general purpose format, such as PDF. This all sounds fairly standardized, but most printing pipelines remain proprietary to specific brands and companies, making these specific drivers vast and varied(Huang & Rudolph, *Bluetooth for Programmers* 2005).

Bluetooth is all the rage these days, as the BLE(Bluetooth Low Energy) connectivity model allows users to wirelessly connect devices to one another. An operating system only needs a driver and a bluetooth adapter to allow for connectivity to occur. Mac OS X was the first operating system to support bluetooth way back in 2002, and Microsoft followed soon after by implementing support through a Windows XP update (Lawday, *Implementing Bluetooth in an Embedded Environment* 2001). Earlier versions of XP even required users to download their own adapter drivers. Apple and Microsoft both use their own proprietary bluetooth stack, but open source operating systems like Linux and Android OS use BlueZ and Fluoride respectively. BlueZ was developed by Qualcomm, and is implemented in most Linux based kernels today. Fluoride, also known as Bluedroid, was developed by Broadcom for use in Android based phones, but was used in Linux systems as well.

Conclusion/Wrap Up:

There are several types of devices that an operating system needs to have the capability to interact with. To manage the multitude of different implementations of devices, drivers are used to bridge this interaction. Drivers provide a software interface for the operating system. Through this interface, the operating system can control the devices. Further work for this research would be deeper dives into source code for each classification of drivers.

References:

Aviviano. (n.d.). Network Driver Design Guide - Windows drivers. Retrieved November 14, 2020, from https://docs.microsoft.com/en-us/windows-hardware/drivers/network/

Contributors to Wikimedia projects. (2012b, October 8). *Hard disk drive interface - Wikipedia*. Wikipedia, the Free Encyclopedia; Wikimedia Foundation, Inc. https://en.wikipedia.org/wiki/Hard_disk_drive_interface

Contributors to Wikimedia projects. (2020, November 12). *Parallel ATA - Wikipedia*. Wikipedia, the Free Encyclopedia; Wikimedia Foundation, Inc. https://en.wikipedia.org/wiki/Parallel_ATA

Contributors to Wikimedia projects. (2020a, August 30). *SCSI command - Wikipedia*. Wikipedia, the Free Encyclopedia; Wikimedia Foundation, Inc. https://en.wikipedia.org/wiki/SCSI_command

Contributors to Wikimedia projects. (2012, May 5). *ATA Packet Interface - Wikipedia*. Wikipedia, the Free Encyclopedia; Wikimedia Foundation, Inc. https://en.wikipedia.org/wiki/ATA_Packet_Interface

Contributors to Wikimedia projects. (2020a, August 30). *SCSI command - Wikipedia*. Wikipedia, the Free Encyclopedia; Wikimedia Foundation, Inc. https://en.wikipedia.org/wiki/SCSI_command

Contributors to Wikimedia projects. (2020a, February 18). *Logical block addressing - Wikipedia*. Wikipedia, the Free Encyclopedia; Wikimedia Foundation, Inc. https://en.wikipedia.org/wiki/Logical_block_addressing

Cossu, N. (2011, March 19). *wmi - how to check if hard drive is eide or sata with powershell - Stack Overflow*. Stack Overflow; Stack Overflow.

https://stackoverflow.com/questions/5362199/how-to-check-if-hard-drive-is-eide-or-sata-with-powershell

Corbet, J., Rubini, A., &amp; Kroah-Hartman, G. (n.d.). Linux Device Drivers, 3rd Edition. Retrieved November 14, 2020, from https://www.oreilly.com/library/view/linux-device-drivers/0596005903/ch17.html

Device driver. (2020, November 10). Retrieved November 12, 2020, from https://en.wikipedia.org/wiki/Device_driver

DriversExpert. (2017, February 27). What are Device Drivers and why do we need them? – Drivers.com updates. Retrieved November 15, 2020, from https://www.drivers.com/update/drivers-news/what-device-drivers/

Garney, J., & Lueker, J. (2000). *Universal Serial Bus Specification* (Revision 2.0). Compaq Computer Corporation, Hewlett-Packard Company, Intel Corporation, Lucent Technologies Inc., Microsoft Corporation, NEC Corporation, Koninklijke Philips Electronics N.V.

Huang, A., and Rudolph, L. (2005). Bluetooth for Programmers.

Lawday, G. (2001, October 10). Implementing Bluetooth in an Embedded Environment -. Retrieved November 15, 2020, from https://www.eetimes.com/implementing-bluetooth-in-an-embedded-environment lorihollasch. (2018a, March 29). *IOCTL_SCSI_PASS_THROUGH (ntddscsi.h) - Windows drivers | Microsoft Docs*. Developer Tools, Technical Documentation and Coding Examples | Microsoft Docs; Microsoft. https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddscsi/ni-ntddscsi-ioctl_scsi_pass_through

lorihollasch. (2018b, March 29). *IOCTL_ATA_PASS_THROUGH (ntddscsi.h) - Windows drivers | Microsoft Docs*. Developer Tools, Technical Documentation and Coding Examples | Microsoft Docs; Microsoft. https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddscsi/ni-ntddscsi-ioctl_ata_pass_through

MIT. (2006). Multics Internet Server. Retrieved November 12, 2020, from http://web.mit.edu/multics-history/source/Multics_Internet_Server/Multics_sources.html

Network Devices. (n.d.). Retrieved November 14, 2020, from https://www.tutorialspoint.com/communication_technologies/communication_technologies_network_devices.htm

Oney, Walter (2003). *Programming the microsoft windows driver model*. Microsoft.

Peters, T. (2020, September 2). *An Overview of Computer Systems (Part II)*. CS 460: Operating Systems; Travis Peters. https://www.traviswpeters.com/cs460-2020-fall/files/cs460-02-system-overview-p2.pdf

Rouse, M. (2020, January 22). Device Driver. Retrieved November 15, 2020, from https://searchenterprisedesktop.techtarget.com/definition/device-driver

Smith, J. (2014, August 10). *c - How can I identify the protocol used in hard disk? - Stack Overflow*. Stack Overflow; Stack Overflow. https://stackoverflow.com/questions/25185410/how-can-i-identify-the-protocol-used-in-hard-disk

Weber, R. (2008). *SCSI Primary Commands-3* (Revision 1). T10 Technical Committee.

Weber, R. (2013). *ATA/ATAPI Command Set - 3* (Revision 5). T13.

Xerox Palo Alto Research Center (1980). *Alto operating system reference manual.* Retrieved from http://xeroxalto.computerhistory.org/_cd8_/altodocs/.os.press!2.pdf