Zack Chambers

CST 221

January 27, 2019

John Zupan

GitHub Link: <https://github.com/zchambers3/CST-221/tree/master/fileIO>

**File I/O**

**User-Space I/O Software**

*Description*

An interface of libraries and/or library procedures that can be accessed through kernel and/or user that allows the interaction with device drivers. This is a program that will provide ease of interpretation of commands entered by a user. It is similar to when someone whose main spoken language is English, and they are learning to speak Spanish. When they learn to say hello their mind/human interface can recognize what the word is in Spanish and say Hola aloud.

*Purpose*

The purpose is to accurately and efficiently route commands entered by the user out to the location designated by the user or default location designated by the current running program. I/O software mainly uses library procedures whereas spooling systems mainly deal with devices in a program with multiple systems. Sometimes spools are managed by a daemon process in OS. Daemon processes run in the background of an OS and are not in control by a user.

*Example*

An example is the user level I/O library stdio setchar(), getchar(), printf(), and scanf() available in C programming. The getchar() function is used to read a character from a keyboard input. In a C program, we can utilize the getchar() function as shown below. The program will ask the user to enter a character and will return the character that was entered.

#include <stdio.h>

int main () {

char c;

printf("Enter character: ");

c = getchar();

printf("Character entered: ");

putchar(c);

return(0);

}

Output:

Enter character: a

Character entered: a

**Device Independent I/O Software**

*Description*

Although it is difficult to write fully independent device software, we can create some modules common to all devices. This part serves as a framework for how different devices interact with the system as a whole. The device-independent code does most of the functionality, but not necessarily most of the code, as many drivers can all do the same thing in a slightly different way thanks to somewhat different controllers.

*Purpose*

The function of device-independent software is to control and operate the I/O functions of all basic devices and provide a functional interface at the user-level. Some of these functions include error reporting and uniform interfacing for device drivers. An error report might be caused by a process that asks for something impossible or accessing a damaged part of a disk. Uniform interfacing can make all devices look the exactly the same or similar by normalizing the driver interface.

*Example*

File systems in Linux include folders for each device kept on the system. These include "/dev / psaux" or "/dev / ttyS0" and the specific device information is stored in folders when the devices are installed. Device files allow the user to access hardware. If you access “cat /dev/dsp > my\_recording” then say something into your computer microphone, then type “cat my\_recording > /dev/dsp” the system should play back what was said in the microphone. Programs that access device files do so by reading and writing to the device to send and retrieve data, and by the use of the I/O control function that configures the device or devices.

**Device Drivers**

*Description*

Device drivers are software modules which are connected to an operating system to handle a specific device. It operates specifically on the devices interface and code contained within that device. Device drivers will also check for hardware status to if a request can be handled. It will block itself and wait for interrupt, or upon waking it will check for I/O errors, pass data to the layer above it and return status to its caller. It is a translator/communicator between an OS and applications that use the devices.

*Purpose*

Device drivers start by accepting the device's independent I/O software command and formatting it to work with the installed driver They also initialize and wake the driver up if it is idle or still needs to be used. The device driver interacts with the device controller to perform the necessary error handling and I/O.

*Example*

The Linux structure means that device drivers are numerous and easy to build. Presume a request reads a block B. If the driver is idle when a query arrives, the request starts immediately. If the driver is already preoccupied with another request, the new request is placed in the queue of pending requests.

**Interrupt Handlers**

*Description*

An interrupt handler, also referred to as an interrupt service routine or ISR, is a software program or, more explicitly, a callback function in an OS or, more specifically, in a device driver whose implementation is triggered by an interrupt.

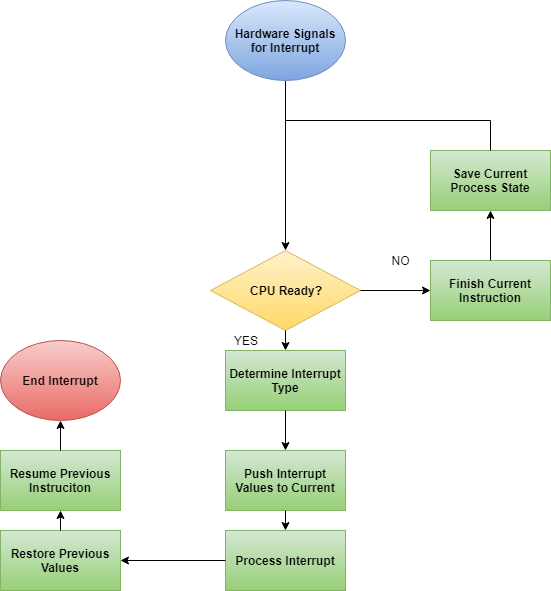
*Purpose*

When the interruption occurs, the interrupt procedure does everything it needs to handle the interrupt, updates data structures and awakens the process waiting for an interrupt.

*Example*

Hardware interrupts occur within devices such as keyboards, clocks, disk, network cards or a mouse. Depending on the type of interrupt request the CPU will then send the request to the corresponding hardware driver. Hardware drivers are subroutines that occur within the kernel instead of a separate process. Once one I/O completes a routine, the next item in the queue is delivered to the device. Software interrupts are handled much like hardware interrupts. The only difference is that they are only generated by processes that are running in that moment in time. Avoiding the hardware signaling step. Software interrupts do interrupt the flow of the calling code. It will cause the CPU to interrupt and direct its attention to the code the software is running. The CPU will take the state that it was in prior to the interrupt and put it in a stack where it will return to after it has handled the interrupt. Software interrupts might stop a running program, an idle process or even the kernel code that might be handling another interrupt.

Hardware Interrupt Flowchart



*Hardware Interrupt Flowchart Summary*

After the hardware task is completed, a signal is received by the interrupt handler. The interrupt handler will then format the output of the data format and a send the information to the CPU to begin the interrupt process. The necessary queues and tables are constructed by the interrupt handler for the CPU to manage the process. The process that the CPU was in prior to the interrupt will be set aside in a stack while the current interrupt is handled. Once the current process is done the program that was interrupted will then return to its original state.

**Keyboard Function**

*Theory of operations*

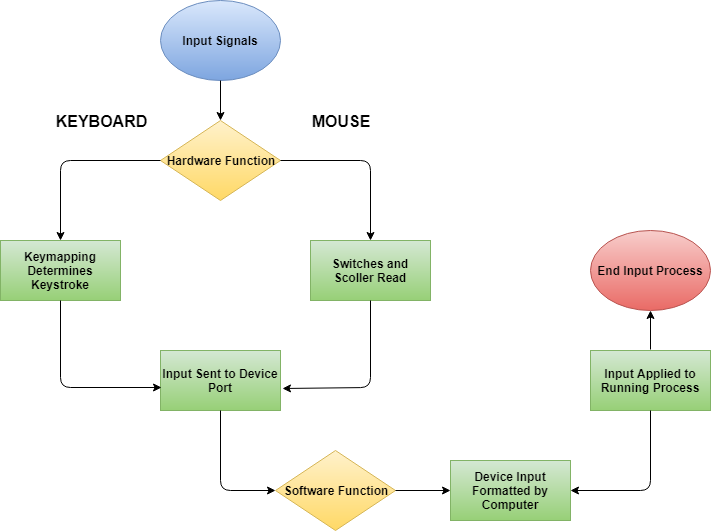
A computer keyboard is an input device used by pressing buttons or keys for entering characters and functions into the computer system. It is the primary tool for entering text. Typically, a keyboard contains keys for individual letters, numbers and special characters and keys for specific functions. Keyboards have a wide array of uses. For example, a person may use a keyboard for writing a document, access menus within their OS through various keystroke shortcuts and maybe even play games. They are designed to help the user finish tasks in lightning speed. One can also assign specific commands to the function keys at the top of the keyboard or even utilize the control keys to control the screen of a computer. PC keyboards actually use a scan code then the keyboard driver will convert the scan code to ASCII, which is a 7-bit character set that contains 128 characters. Scan code is the data that a keyboard actually sends to the CPU to let the CPU know what character has been pressed. A popular interrupt that used quite often to lock your computer is the ctrl + alt + delete command. It will immediately interrupt any process the user is performing and allow the user to either restart the computer or access the task manager program.

**Mouse Function**

*Theory of operation*

So, while the keyboard is set to type, the mouse is designed to point and click around a GUI, such as Windows or the Mac OS, as well as the many applications running on those operating systems. The mouse, a small terror to some and now a device on your desk, assists users in a multitude of tasks while operating a computer. The computer receives the signal from a mouse via WIFI or a cable, both options utilizing a USB port. An optical mouse uses a light on the bottom that bounces off the desk and into a photocell. As the mouse is pushed, the pattern of the light changes and the hardware in the mouse takes this information and figures out where your hand is moving. The scroll wheel mechanism at the top of the mouse allows users to move up and down on pages, the left switch usually selects an item on the computer and the right switch will display several different actions items for the user to execute. When playing computer games, it is extremely beneficial using a mouse that has a high DPI (dots per inch) that translates to more reads per second. This means the light being reflected will result in more frequent position updates, which essentially makes the mouse move faster. However, the requirements of the mouse depend highly on the task at hand. If a user is using a computer mainly to write documents, then maybe a mouse with a high DPI is not needed. Interrupts never occur while operating a mouse as you can click a mouse multiple times without it having any type of effect on the current running program.

Mouse and Keyboard Flowchart



**References**

CodesCracker.com. (2018). *OS Input / Output Software Layers*. Retrieved from <https://codescracker.com/operating-system/input-output-software-layers.htm>

Michael, K. (2010). *The linux programming interface : a linux and unix system programming handbook*. Retrieved from <https://ebookcentral-proquest-com.lopes.idm.oclc.org>

Ray, J. (1999). *Sams Teach Yourself Linux in 10 Minutes*. Indianapolis, IN: Pearson Education, Inc. Retrieved from <https://lopes.idm.oclc.org/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=12198&site=ehost-live&scope=site>

Tlpd.org. (2018). *Linux Filesystem Hierarchy.* Retrieved from <http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/dev.html>

Tyson, J. Wilson, T. (2018). *How computer keyboards work.* Retrieved from <https://computer.howstuffworks.com/keyboard.htm>

Woodford, Chris. (2017). *Computer Mice*. Retrieved from <http://www.explainthatstuff.com/computermouse.html>