Zack Chambers

CST 221

January 27, 2019

John Zupan

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

**File I/O**

**User I/O Software**

*Description*

User I/O is considered the most important of I/O software. This includes programs running outside of the kernel and operating system. This is usually a program that retrieves or interprets user commands, including libraries linked to these user routines.

*Purpose*

User I/O’s purpose is to correctly interpret and route user commands to the correct location. This layer formally calls the subsystems to I/O and correctly formats the data.

*Example*

An example of I/O software would be the terminal. User commands must be correctly formatted to be submitted and passed to the subsystems for proper execution once they are done. These consist of “open [file]” and “cd [folder]”. The terminal itself is the User Software that provides direct I/O with many different subsystems.

**Device Independent I/O Software**

*Description*

This layer consists of general functions and management systems for the system drivers. 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*

This layer handles buffering or user input and device output, since both are handled by abstraction. This layer also handles error reporting, device allocation and a uniform interface for devices.

*Example*

The Linux filesystem includes folders for each device kept on the overall system. These include "/dev / psaux" or "/dev / ttyS0" and the specific device information is subsequently stored in these folders when the devices are installed.

**Device Drivers**

*Description*

Device drivers house specific device code that controls the attached device's hardware. These programs are fully run in the kernel mode unless the device is programmed differently. Device drivers are classified into two main categories, block drivers and character drivers. The device driver portion of the OS knows the controller's characteristics.

*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. This layer interprets any interruptions and sends back the data to the chain.

*Example*

The Linux structure means that device drivers are numerous and easy to build. Each is written to interact with the Linux kernel and inserted as the majority of device drivers are implemented. Keyboards, disks or printers are all device drivers and have their own specific code to enable them to work and communicate with any device port.

**Interrupt Handlers**

*Description*

Interrupt handlers are the code that handles stopping current processes when output is generated and must be restored to the chain. Interrupts are sent as signals that allow the CPU to determine when the interrupt is being handled. In other words, when the process is ready, it is up to the scheduler to decide when to run it.

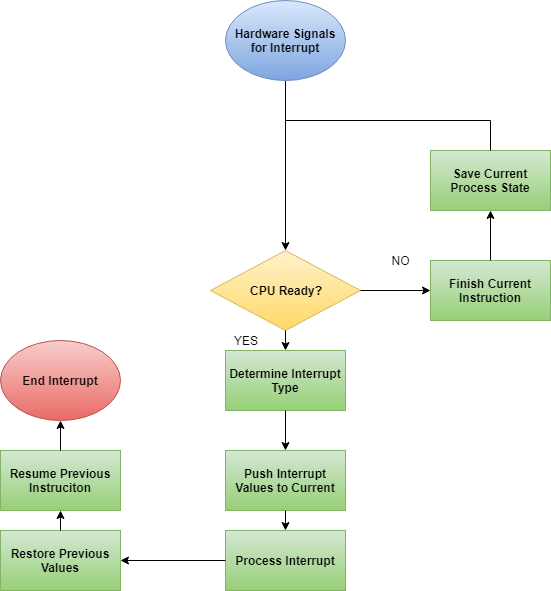
*Purpose*

Interrupt handlers initially activate the saving of the current process and then determine the type of interrupt based on the interrupt vector. The hardware interrupt also has to set the context and stack for the routine. This is all done to ensure that process A can be completed and that the interrupted process B also does not potentially lose any data.

*Example*

Modern Linux systems can handle interrupts from multiple devices in a programmable manner using APICs, which stand for advanced programmable interrupt controllers.

Flowchart



*Flowchart Summary*

Once the hardware has completed its task, the interrupt handler receives a signal. The interrupt handler then determines the output data format and sends a signal to the CPU for the interrupt processing. The interrupt handler then establishes all required tables, queues, etc. that the interrupt requires and gets ready for the CPU to start managing the process. The CPU saves the current execution status of the current process and then manages the interrupt. Upon completion of the interrupt process, the previous program status is returned and thus the execution continues.

**Keyboard Function**

*Theory of operations*

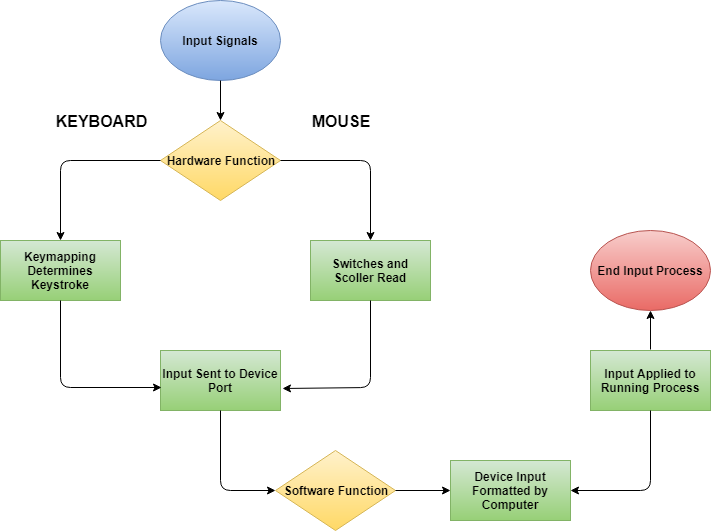
A keyboard is considered to only be an input device. The keys a user might press are the hardware, the key matrix that is situated under the keyboard can be regarded as the I/O layer for the user. This interprets keystrokes with switches that complete a circuit by pressing down the key. The independent software for the keyboard device then interprets the meaning of the keystroke by comparing it with the keymapping installed with the individual keyboard. The keystrokes are then passed to the driver of the keyboard device that communicates with the CPU and other applications, the kernel or the user. A keyboard has at least one type of interrupt function, usually the ctrl+alt+delete, which pauses any current process immediately and allows the keyboard to enter new data.

**Mouse Function**

*Theory of operation*

A typical mouse contains some parts considered part of the user input layer, including the left and right buttons, the scroll wheel and the analog movement device. When any of these processes are prompted, the chip reads the movements or clicks and transmits them to a USB port via a cable or a wireless connection. In essence, this function is the device independent layer. The computer then contains a device-based way to interpret the mouse input and then directly translates it into data to be used by the current running application. To my knowledge and understanding, a mouse has no interrupt function, since no matter how much you click, nothing at all occurs when a program does not respond directly.

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>