Michael Weaver

CST – 221

John Zupan

2/18/18

GitHub Link: <https://github.com/battousairurik/CST-221>

**File I/O**

User I/O Software

*Description*

User I/O is considered the top level of I/O software. This consists of programs which run outside the kernel and outside of the OS. This generally consists of programs which retrieve or interpret user commands, including libraries linked to these user routines.

*Purpose*

The purpose of User I/O is to properly interpret user commands and route them to the correct location. This layer formally makes the I/O call to the subsystems and formats the data properly.

*Example*

A Linux example of this would be the terminal. User commands must be properly formatted to be submitted and once they are done so they are passed to the subsystems to execute properly. These include “open [file]” and “cd [folder]”. The terminal itself is the User Software which allows direct I/O with the numerous subsystems.

Device Independent I/O Software

*Description*

This layer consists of generic functions and management systems for the device drivers contained within a system. This portion acts as the framework for how various devices interact with the overall system.

*Purpose*

This layer handles buffering, or interpreting user input and device output, given that both are handled through abstraction. Error reporting, device allocation, and uniform interfacing for devices are all handled by this layer as well.

*Example*

The Linux filesystem contains folders for each device maintained on the overarching system. These include “/dev/psaux” or “/dev/ttyS0” and the specific device information is then stored within these folders as the devices are installed on the system.

Device Drivers

*Description*

Device drivers house device specific code which controls the hardware of the attached device. These programs are run fully in kernel mode, unless the device is specifically programmed otherwise. Device drivers fall into two main categories, block-device drivers, and character-device drivers.

*Purpose*

Device drivers begin by accepting the command passed through the device independent I/O software and format it to work with the installed driver. They also initialize and wake up the driver if it has become idle or has yet to be used. This layer also interprets any interrupts and sends the data back up the chain.

*Example*

The structure of Linux causes device drivers to be both numerous and easily built. Each are written to interact with the Linux kernel and be inserted as most device drivers are. Keyboards, disks, or printers are all considered device drivers and contain their own specific code to allow them to function and interact with any device port.

Interrupt Handlers

*Description*

Interrupt handlers are the code which handles stopping current processes when output has been generated and needs to be passed back up the chain. Interrupts are sent as signals, allowing the CPU to determine when the interrupt is handled.

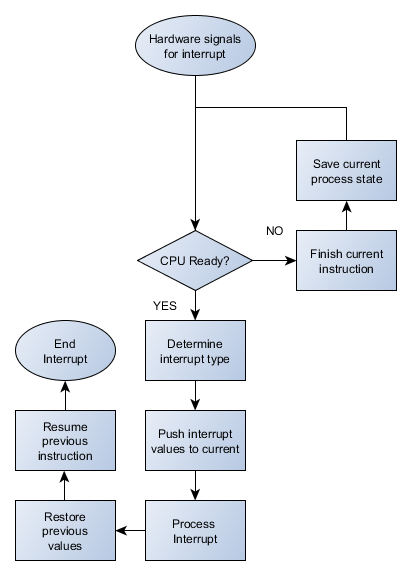
*Purpose*

Interrupt handlers first trigger the save of the current running process, they then determine the type of interrupt based on the interrupt vector. The interrupt hardware must also set up context and stack for the routine. All of this is done to ensure that process A can be executed to completion and the interrupted process B does not lose any data.

*Example*

Modern Linux systems use APIC’s, or advanced programmable interrupt controllers, which can handle interrupts from multiple devices in a programmable way.

Flowchart



*Summary of flowchart*

After the hardware has finished its task, a signal is sent back to the interrupt handler. The interrupt handler then determines the format of the output data while sending a signal to the CPU to process the interrupt. The interrupt handler then sets up all necessary tables, queue’s, etc. that the interrupt requires and waits for the CPU to begin handling the process. The CPU saves the current state of execution of the current process, then handles the interrupt process. Once the interrupt process is finished, the previous program state is returned, and execution continued.

Keyboard Function

*Theory of operations*

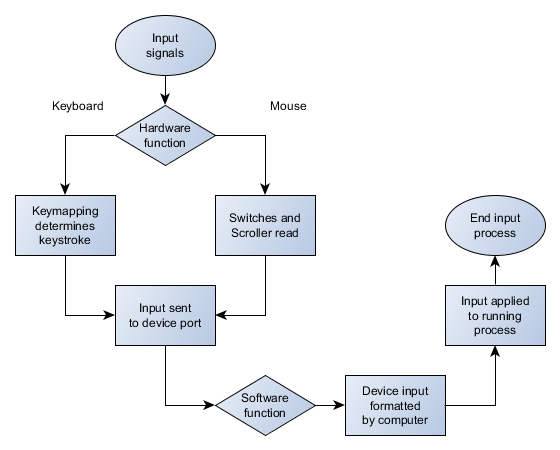
A keyboard is strictly an input device. The keys that we press are the hardware, the key matrix that is located beneath the keyboard can be considered the user I/O layer. This interprets keystrokes using switches, where a circuit is completed by pressing the key down. The keyboards device independent software then interprets the meaning of the keystroke by comparing it to the keymapping installed based on the individual keyboard. The keystrokes are then passed to the keyboard device driver which communicates with the CPU and other running applications, Kernel, or user. A keyboard has at least one type of interrupt function, typically the ctrl + alt + delete which will immediately pause any current process and allow the keyboard to input new data.

Mouse Function

*Theory of operation*

A typical mouse contains a few parts that are considered to be part of the user input layer, these include the right and left buttons, the scroll wheel and the analog movement device. When any of these functions are triggered the built-in computer, chip reads the movements or clicks and transmits them through a cable or over a wireless connection to a USB port. This function is essentially the device independent layer. The computer then contains a device-based manner with which it reads the mouse input, then translates it into data to be used by whatever application is up and running. To my knowledge a mouse does not have any interrupt function, given that when a program does not respond, no matter how much you click nothing will occur.

Flowchart for Mouse and Keyboard



**References**

Gottlieb, Allan. (2003). *Lecture #13: Principles of I/O Software.* Retrieved from <https://cs.nyu.edu/courses/spring03/V22.0202-002/lecture-13.html>

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

CS 446/646. (2005). *Principles of Operating Systems.* Retrieved from <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&cad=rja&uact=8&ved=0ahUKEwjixa6Mva7ZAhWkTd8KHSF7DLsQFgheMAQ&url=http%3A%2F%2Fdoursat.free.fr%2Fdocs%2FCS446_F05%2FCS446_F05_5_IO2_Software.pdf&usg=AOvVaw3_mPUmknVoB1SafjDPNDF9>

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>