WRITING TOPIC TWO

I/O AND PROVIDED FUNCTIONALITY

November 11, 2016

Steven Silvers
Oregon State University
CS 444 Operating Systems II

I. Provided Functionality in Linux

The Linux operating system provides a lot of useful built-in functionality including popular data structures, common algorithms and various security and cryptographic features[1]. Data structures are seen commonly throughout the Linux kernel and are defined in header files in the include directory. Things like sockets, network devices, files and PCI buses are all represented and described by data structures built in to the Linux kernel [2]. Without these data structures in place the kernel would not be able to interact with the various devices properly. Linked Lists in the Linux kernel are implemented differently from the traditional linked list where the data field is part of the node structure. In the kernel version, there is a data type called "list_head" defined in the types.h file[1]. This data type is used to embed the list into the data, as opposed to the method mentioned before where the data is embedded into the list. This method combined with how the C language can be manipulated makes the kernel's implementation of linked lists completely type safe.

The next major category in Linux provided functionality is the collection of built-in algorithms. One of the collections of algorithms provided by Linux are the elevator algorithms. The elevator algorithms are used to handle request servicing by deciding what order requests should be serviced in a way that reduces overall service time[1]. Almost all elevator algorithms perform some of the same tasks, such as merging and sorting as well as a way to handle aging of requests to prevent any one request from sitting around and not being serviced[1]. Other common provided Linux algorithms include red black trees, binary searches and a variety of sorting methods. Red black trees are a type of binary search tree that is self-balancing and are commonly used in searches as well as virtual memory management[3]. Red black trees are similar to AVL trees, but in red black trees lookup speed as been sacrificed to improve insert and delete worst case performance[3]. Linux also provides its own version of the popular binary search. In order to function it requires the array that it is acting upon to already be sorted in ascending order. The Linux binary search is most commonly used for interrupt handling within the kernel.

The Linux kernel has a large amount of provided functionality in regards to cryptography, including the Linux Crypto API introduced in version 2.5.45[4]. In newer versions of Linux this API has been expanded to include block ciphers and hash functions. This API allows users to add encryption to their own custom devices within the Linux kernel. For example, in the third kernel project for this class we are building our own RAM disk driver and using the Linux crypto API to add encryption functionality to the driver. Since everything in Linux is represented as a file, it is possible to encrypt your entire file system. This makes your file system impossible to read without the encryption keys, which could be stored on a small flash drive to keep you system secure[5].

II. I/O

I/O is one of the most fundamental pieces of functionality in all operating systems. Without I/O things like the keyboard, mouse, monitor wouldn't exist and would be near impossible for a user to interact with a computer at all. Block I/O is used when the data coming through the I/O stream can be moved as blocks using a large buffer to hold the data. Block I/O is most commonly used reading disk drives and other memory storage systems. Character I/O is when each piece of data comes through character by character and is not lumped together like in the block I/O. An example of character I/O would be a keyboard or a terminal. Currently my text editor is waiting on input from my keyboard and as soon as I press a key, that character is sent to the text editor which then displays the new character back to me on the terminal.

Drivers are the software layer of peripheral devices that enable the I/O device to communicate with the kernel properly. Drivers are typically developed by the third parties who also developed the I/O device in question. Windows operating system handles device drivers very well, because Windows is a closed operating system it keeps drivers separated from the Windows kernel and does not include any pre-installed drivers in the kernel. On the other hand, Linux is open source and treats drivers as modules that can be loaded and unloaded into the kernel dynamically[6]. Windows provides a binary interface so that drivers can interact with the kernel, keeping a healthy level of separation. This separation allows for very well executed plug and play functionality for I/O devices with the Windows operating system. Due to the open source nature of Linux, there isn't great need for the layer of separation between drivers and the kernel like in Windows, since the source code for Linux is readily available

to everyone it makes sense to go ahead and integrate drivers as modules within the kernel. The biggest similarity between Linux and Windows in regards to drivers is that they both use event-driven APIs, meaning the driver code is only called in response to something happening[6].

Linux has many schedulers built in and can change between them at runtime. These include the NOOP scheduler, LOOK, C-LOOK and the Completely Fair Queuing which is typically the default I/O scheduler for Linux[1]. FreeBSD uses C-LOOK as its default scheduling algorithm which is designed to maximize throughput[7]. Windows uses their own priority based system where the task scheduler gives each I/O task a priority level and then executes tasks based on priority. Linux has the edge here, having the ability to implement multiple schedulers makes Linux better suited to adopt well to its needed roll based on the expected I/O load and type. For example, if you plan to use Linux to run your server, you could use an I/O scheduler that focuses on block I/O and not worry about character I/O from things like keyboards and monitors, since those will rarely be used. If you want to run Linux as a user desktop, you could then again select a scheduler that is better suited to those kinds of tasks. In the Windows operating system you would have to completely change what version of Windows you are running to get the same kinds of benefits.

FreeBSD and Linux have the most similarities when it comes to device I/O, this is mostly due to both being based on UNIX and both being open source. This allows both operating systems to be highly customizable in regards to how the user would like them to run, whereas standard Windows I/O is optimized to be a user desktop workspace that works well with third party drivers and devices.

REFERENCES

[1] K. McGrath, *Operating Systems*. Top Hat, 2016.

[2] D. A. Rusling, *The Linux Kernel*. 0.8-3 ed., 1999.

[3] R. Landley, "Red-black trees (rbtree) in linux." https://www.kernel.org/doc/Documentation/rbtree.txt. Accessed: 11-11-2016.

[4] "Kernel crypto api interface specification." http://www.chronox.de/crypto-API/crypto/intro.html. Accessed: 11-11-2016.

[5] "Linux and cryptography." http://www.antipope.org/charlie/old/linux/shopper/167.crypto.html. Accessed: 11-11-2016.

[6] "Linux vs. windows device driver model: architecture, apis and build environment comparison." http://xmodulo.com/linux-vs-windows-device-driver-model.html. Accessed: 11-11-2016.

[7] G. V. N.-N. Marshall Kirk McMusick, *Design and Implementation of the FreeBSD Operating System*. Addison-Wesley, 2nd ed., 2015.