Drew Pulliam – DTP180003

CS3377.0W1 - All Essays

**Week 2 Essay**

**Reflection on “The History of UNIX and GNU–Linux”**

This chapter gives a brief overview of UNIX and how Linux was started. UNIX was first made for researchers and used primarily in universities. Early on it was inexpensive and usable by large amounts of people. It later became popular in industry which resulted in it becoming more specialized and proprietary. This made it much harder to gain access to UNIX for the average person. The GNU Project was created with the hope to give “free software” to people. The software would typically still cost money but would be available to modify and expand upon. This project was used as groundwork for Linux later.

Linux was designed to be as efficient and powerful as possible, in contrast to other operating systems like UNIX such as MINIX. It was designed to be able to easily accommodate many users using the same physical hardware at the same time. This was accomplished in several ways, one of them being the ability to use different shells simultaneously. Linux uses different shells depending on what the user wants to accomplish. With the capability for multiple users, they needed to add capability for multiple shells as well. As a result, multiple users can all be using the same machine and they can all use different individual shells as well.

Some of the benefits of using Linux over other operating systems that are not “free” is that it saves money for companies and consumers. A computer manufacture that makes Windows laptops must pay Microsoft on a per-unit basis for Windows licenses. Linux does not require this which will save obviously save money for companies and will also result in them being able to reduce prices for consumers. Companies are seeing the value of having Linux as an operating system, and they are paying their employees to help contribute to the continued development and updating of Linux.

**Week 3 Essay**

**Reflection on APUE 1.1-1.6**

These sections give a brief overview of the UNIX operating system. The UNIX architecture is all built around the kernel. The kernel is the software that controls the actual computer hardware, and other programs simply request access to this hardware through the kernel. The way for programs to request resources from the kernel is through system calls. These system calls can be made in several ways, most commonly through the shell. The shell is a command line interface that allows the user to access the kernel. Technically Linux is just the kernel, and GNU is the rest of the operating system, but it is typically all referred to as simply Linux.

Logging into a UNIX system is a simple as entering your username and password. The system will then check your password and username against a saved and encrypted file to make sure that they match and allow you to log on. There are then several different options of shells that can be used, most commonly Bourne, C, Korn, and TENEX C. There are many similarities between these shells, specifically Korn is considered a successor to Bourne and TENEX C is a successor to C.

The UNIX filesystem is simple, everything is a descendent of the root directory, which is denoted by simply “/”. When using a shell, there is a current working directory, and the user can specify file paths starting at the current working directory (relative pathname) or starting at root (absolute pathname).

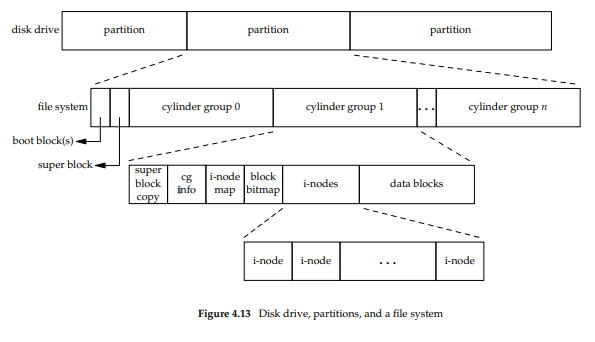
Standard Input, Output, and Error are three different options that all shells have for programs running in them. If none are specified they all default to the command line, but users have the option to specify what they want these to be (typically text files). This allows a user to save a standard input as a file and not be required to retype it every time they want to run a specific program. It can also allow for easier access to program output and possible error messages.

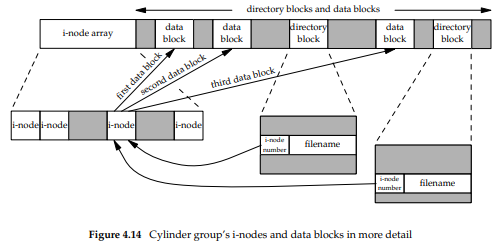
**Week 5 Essay**

**APUE 4.14 File Systems tutorial**

There are several different implementations of file systems in UNIX. Solaris (which supports different types of disk file systems), UFS, PCFS, and HSFS (reads CD file systems). Each file system has different features.

Disk drives are divided into partitions. Each partition may contain its own file system (Figure 4.13).

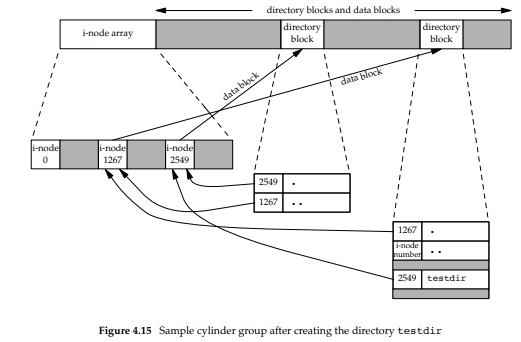




Examining i-nodes and data blocks in more detail results in Figure 4.14.

* Two directory entries point to the same i-node. Every i-node has a count of the number of links that point to it, and it can only be deleted when the count is at 0 (nothing is pointing to the i-node).
* Another type of link is a symbolic link. The actual contents of the file store the name of the file that the link points to. This means that a symbolic link is actually just a file that contains the name of the file it links to.
* The i-node contains info about the file it is associated with. File type, access permissions, size, and more. The only information stored in the directory entry is the filename and the i-node number (basically it just points to the i-node for all the information).
* The i-node number in the directory will always point to an i-node in the same file system. The stops it from pointing to i-nodes in different file systems.
* If you wish to rename a file without changing file systems, the actual file does not need to be moved. Instead all you need to do is add a new directory entry that points to the same i-node and unlink the old directory entry. (this works because the filename is stored in the directory entry not the i-node)

Figure 4.15 shows the result of making a new directory inside the working directory (mkdir testdir). The i-node 2549 is a directory with a link count of 2. Leaf directories always have link counts of 2. This is because the two links are the name of the directory and the entry for dot.



**Week 6 Essay**

**Reflection on APUE 3.9-3.12**

Section 3.9 starts off with a simple program used to copy files using read/write. It then goes a bit more in depth explaining how BUFFSIZE changes how fast the copy can be completed, and it suggests a size of 4096 is usually sufficient as larger sizes have diminishing returns.

Section 3.10 describes how the kernel represents open files. The three data structures used are: 1. A process table with entries for each open process, and file descriptors inside the process entries. 2. A file table of all open files, including status flags for each file. 3. A v-node structure contains information about each file. All 3 of these tables have been part of UNIX for a long time, and they are very important regarding how files are shared with different applications on the UNIX machine.

Section 3.11 starts by explaining how appending to a file by two different processes simultaneously could create problems. If two processes open a file and determine that the last line is line 500 for example, but then process A writes 5 more lines to the file, process B might still think the end is line 500. This could result in process B overwriting process A’s data. The way UNIX solves this is by making an O\_APPEND flag. This flag will allow all processes using a file to append to that file without fear of other processes overwriting their data. This problem can also happen when two processes try to create a file with the same name, which could erase the first one created. UNIX again solves this by combining the check for files with the same name with creating a file. When processes try and do these actions separately it could create overwritten files.

Section 3.12 is about dup and dup2 functions. They are used to duplicate file descriptors. Dup simply creates the lowest number available file descriptor, and dup2 takes two arguments, which allows you to specify what number descriptor you would like to use as the duplicate.

**Week 10 Essay**

**Reflection on Internet Protocol Suite**

The internet protocol suite is built on TCP/IP protocols (Transmission Control Protocol and Internet Protocol respectively). It was developed by the United States Department of Defense through DARPA. The first early research started with DARPA’s ARPANET in 1969. Later DARPA started working on other data transmission technologies. Eventually they started using a common internetwork protocol instead of relying on the network for reliability.

Initially the development of the internet protocol was not layered enough, it was built more as a monolithic design which would have created scaling issues. Eventually it was switched into a better design, specifically the TCP/IP protocols which divided the work very well and allowed for better scaling. It was so simple that it does not do anything except transmit data from one point to another. All real data manipulation and other useful processes are done at the end points of data transfer. TCP/IP only transfers data, it does not waste time doing anything else.

The protocol was first tested in several different universities. During the testing process the version developed from version 1 to version 4, and version 4 (IPv4) is still used in the internet today (along with the newer version IPv6). In 1975 the first test between Stanford and University College London was successful. Over the next few years, it would continue to be developed in research labs in universities. In 1982 the US Department of Defense declared TCP/IP to be the standard for all military networking. This was the start of mainstream adoption for TCP/IP. It was fully integrated into the existing ARPANET on January 1, 1983.

Over time the simplicity of only sending data and not performing other processes has slightly complicated. The need for firewalls and security have forced more complexity into TCP/IP. Overall, the system is still simple and tries to have as little computation along the way as possible. TCP/IP also still exists using the concept of layering. Specifically, the link layer, internet layer, transport layer, and application layer.

**Week 10 Bonus Essay**

**Reflection on APUE 16.7 and 16.8**

APUE 16.7 is about Out-of-band data. This is a feature that some communication protocols use that allows for high-priority messages to be sent. Out-of-band data basically “jumps to the front of the line” and is sent ahead of any other data that is not already mid transmission. TCP supports out-of-band data (and calls it “urgent” data), but not UDP.

TCP “urgent” data only allows one byte to be sent at a time. You can send urgent data using the MSG\_OOB flag. Technically you can send more than one byte and still use this flag, but only the last byte will be treated as actual urgent data. When urgent data is received there is a SIGURG signal.

TCP also supports “urgent mark”. This is the point in a data stream where urgent data goes. Using this we can send data and specify a specific byte to be urgent (not always the last byte like the default). To do this use the SO\_OOBINLINE socket option.

APUE 16.8 is about nonblocking and asynchronous I/O. There are several functions that can be blocked normally, such as recv and send. These functions cannot be blocked when the socket is in nonblocking mode. In nonblocking mode these functions will instead simply fail instead of blocking. They will also set errno to EWOULDBLOCK to show that they would normally be blocked but can not because the socket is in nonblocking mode.

There is a general support for asynchronous I/O using the Single UNIX specification, and also a separate non-standardized method using the socket mechanism. Socket-based asynchronous I/O can arrange to send a SIGIO signal when ready to read data or when space is open in write queue.

There are two steps to enable asynchronous I/O. First step is to establish socket ownership. Second is to tell the socket to signal when I/O operations won’t block. There are several ways to get socket ownership, one of which is to use F\_SETOWN command. To inform the socket to signal correctly we can use the FIOASYNC command.