Linnaeus University

Faculty of Technology – Department of Computer Science

1DV512 – Operating Systems
Individual Assignment1

Assignment Number: 1
Student Name: Mohammadali Rashidfarokhi
Student Email: mr223jp@student.lnu.se
Course Code: 1DV512
Semester: Autumn 2020

Operating Systems Individual Assignment1

1. Which programming languages were used to implement the Unix kernel? Why?

First and foremost, the idea of developing Unix was to give the ability to developers to use this platform for creating different software for different systems.
Unix has a fame of being portable. That is this system has used C programming language as its second language to enable itself to operate on different platforms.
Moving on, the purpose behind using C programming language for Unix was to advance the code from assembly to elevated level language.
Initially, assembly language was used for Unix. However, it has been mentioned before, Unix moved to C programming for making it clearer. That is, the same function will take a place but fewer lines of codes.
All in all, the assembly code contains 1,000 lines of code. Consequently, it can be shortened to 200 for efficiency and 800 for better performance of hardware functions that are not possible to carry out in using C. On the other hand the C code contains 10,000 lines of assembly code.

2. Is the Unix kernel a complete operating system? How would you compare these concepts?

The Unix operating system is considered to be a thoroughgoing operating system. Considering the following points, it will support the idea of why this operating system is outright.
The Unix system is not an ordinary operating system. What I mean by this is that it holds many libraries and utilities and most importantly, it has a master's program called the kernel.
Since the Unix operation system has differentiated itself from other predecessors, by stating that it has been the first portable operating system, this will allow this system to be used on various platforms.
The idea of Unix was not to be considered as portable. However, after a while, this idea has been changed and Unix started to become a portable system that can perform multi-tasking and having more capabilities.
Some aspects have made this operating system special and complete that are plain text, file system, etc.

- **Plain text**:

It is a string that contains only the printable characters. Moreover, it can also contain some special control characters such as the beginning of the new line and tab.

- **File system**:

Unix would not be able to operate without having a file system. That is, the file system is the central component of this operating system. Moving on, it has been one of the first parts that were used in the initial experiment's version of Unix. Briefly covering the functionality of the file system, it can be used to provide data about the storage and retrieval as well.

❖ **The distinguishes between Unix kernel and complete operation system:**
Firstly, I will cover a short introduction about these two topics then I will compare them in detail.

- **Operating system**:

it is considered as a system program that will be loaded first when the system has been turned on and it will create interface between user and computer.

- **Kernel**:

As I have mentioned earlier, this system program is the central component of an operating system. Also, it is responsible for conversion user command into machine language.

- **Operating system features**:

First and foremost, as I have mentioned above, the operating system creates interface between user and hardware. Moving on, its existence is critical since all the systems need an operating system to be able to run. When it comes to executing, one of the main concerns is the protection that eventually the operating system will erase this concern by providing security and protection. Apart from being the first program running when a PC has booted up, it includes single and multiuser OS and real-time OS.

- **Kernel**:

Unlike the operating system, a kernel is system software that exists as one of the operating system parts. It brings about the interface between application and hardware. Just like the operating system its existence is important since all operating system requires the kernel to be launched. Moving on, it only has two kinds that are monolithic and micro. Its functionality can be summarized as the management of the memory along with the disk, process, and task.

3. If you decided to re-implement the Unix kernel using the modern programming languages and tools, which language(s) would you select and why

Since initially assembly language was used to develop Unix kernel then it was updated by using C language, I will choose C as a modern language for using Unix kernel.

Since this language can work with the assembly language, it will make perfect sense to use this language. Moreover, C was developed to take the Unix kernel operating system to the next level and make this system more efficient and keeping the same functionality with the difference of writing fewer lines of codes. Also, GNU license, which was used in Linux, most of its components are written in C.

However, the usage of C is not just for the Unix kernel. That is, C is still one of the most useful and powerful languages. Specifically, its most famous usage can be seen in databases such as Oracle and MYSQL. Also, it is the same for Microsoft Windows and Linux. What I mean by this is that they are both mostly written in C and using assembly as well.

4. Are new running processes (programs) started from scratch in Unix? What is their relationship to the already running processes?

When it comes to making new processes, the fork system call will become noticeable. What I mean by this is that, first and foremost, the new processes will not start from scratch. Consequently, it is basically a copy version of the original process that is referred to as the parent, and the copy version is referred to as the child. One distinguishes between parent and child system calls is that sharing of primary memory is not visible.

Regarding the files which were running before the fork system call, they will be shared flawlessly after this system call. The processes can pick their own fate which occurs by Informing their part in the relationship. However, the parent is always standing by to get the latest status of its children.

Apart from the fork system call, there is another one that will match and interact with fork smoothly which is called the exec system call. To be more specific, a file can exec from the process. Consequently, it can be formed by exchanging information segments and present text of the process that is meant to be for a new file. It worthy of mentioning that the elderly segments will be lost. No harm will take place by using exec. That is, using the exec system call, it will not result in the transformation of the process.
Also, the files can be accessed after exec because the files that were launched before calling this system call will remain open after the call.
Lastly, if the program sees the necessity of retrieving the control after using the exec system call for a second program, it should fork a child process. To be more specific, in the second program, having the child exec, the parent should stand by for the child this procedure is known as “call”.

5. What are main classes of input-output devices supported by Unix? To which class would a Solid-State Drive (SSD) belong? How about a Brain-Computer Interface (BCI)?

The input-output devices are divided into two main classes that are block I/O and character I/O systems. When it comes to the devices, they are categorized by major number, minor number, and class. The common point about each class is that as an entry point there is an array that is pointing into the device drivers. Each one of them has a specific rule but the noticeable one is for the major number which will index the array when calling a specific device driver.

Regarding the SSD, it is considered as a solid-state storage device that makes use of an integrated circuit to save information tenaciously. Moving on, it will also use flash memory and make this flash memory as the backup storage. When it comes to functionality, SSD can use the old-school HHD interfaces or they can make use of the newer interface versions as well. However, despite the old school HHD drives, the SSD is fixed, and no parts can be moved. Lastly, since SSD is a part of the hard drive and it has a fixed size, it belongs to the block I/O classes.

A brain-computer interface (BCI) also known as a neural control interface (NCI), functions as a straight communication pathway between enhanced or wired brain also it can be communicating with an external device. Moving on, BCI varies from neuromodulation in which it let to create a flow for bidirectional information. Making it short, BCI is mainly a mean which will make the interaction between user and computer possible. The BCI is belong to the character I/O system due to the reason that its size is fixed, and it is a machine learning type.

6. Does the Unix file system design use disks C: and D: (as in DOS or Windows)? Can a Unix system have several disk drives attached at the same time?

When it comes to disks like C or D, MS-DOS will become noticeable. That is, the initial part of an MS-DOS file name following by a colon is a string that recognizes some hardware devices. Moving on, C: stands as a primary hard disk in the operating system. It will point out to a special port address in the device table. On the other hand, the Unix file system is considered a method for organizing and efficiently saving data. Apart from having several important features, the files will eventually be saved and ordered into indexes. Therefore, the Unix file system will not use the same path as the DOS or windows.

One of the examples of the path in MS-DOS will be like the following:

path=c:\windows\command

On the other hand, representing names in the Unix is in the regular file-system name. One of the major differences in Unix is that there is no clear separation of the device portion. The name of the path is not based on the device. Consequently, Unix will use a mount table that will trigger the prefixes of pathnames. It is also worthy of mentioning that when the Unix seek for the name in the file-system directory, it will go for it by using major and minor device number.

One of the examples of the path in Unix will be like the following:

/usr/local/bin:/usr/bin:/sbin:/usr/sbin:.

There has been a serious issue regarding the size of the hard disks since they were considerably bigger than the operating systems can handle. Consequently, the recent models can support splitting a single physical disk and break it down into virtual disks or partitions. However, in the PC, there is no need to be concerned about low disk space.

Unix can have plenty of virtual boxes that are different from each other. Also, sometimes the home directory can be spread between two or three dividers. The reason for this tactic is that launching apps tend to leave log files and temp files behind. Therefore, they can fill up the disk eventually. Lastly, it is critical to have enough space on Unix otherwise Unix will behave weakly and it requires an expert to retrieve it.

7. What is an I-node? What are its contents and their purpose? Are I-nodes used for implementation of directories (if yes, how)?

The definition of the I-node is referred to as a data structure in a Unix-style file system that sets out file system objects known as file and directory. Moving on, the functionality of every single node is saving the attributes, and the disk blocks the locations of each object information. As a result, the files system, specifically the attributes, can contain metadata which simply stands for the last modification of that file. Also, directories are just a list of names that are allocated to the I-nodes, and not surprisingly, each directory has its entry and the same for its parent and each of its children.

As I have mentioned previously the main purpose of I-nodes is saving the data regarding files and directories known as ownership, access mode, and type of the files. For the older versions of the file system implementations, the capacity for the I-nodes is fixed at file system creation.

There are some characteristics of file names and directory implications. That is, I-nodes do not have hard link names but only metadata. Secondly, directories in Unix are just a list of association arrangements. Consequently, each one of them contains an I-node number and file name.

Regarding the usage of I-nodes in the implementation, apart from the fact that a file system requires to save files and they also have directories. The files will be saved in the directories and subdirectories. The reason is that something or a location needs to hold this information so they can be retrieved.
File system in the Linux that has I-node cooperate with directory structures to create a framework to support all the metadata for each file and directory. The purpose of doing this is to make the metadata available for whoever needs this information. It can be used by the kernel, user application, or Linux utilities. Lastly, since the I-nodes hold the data about files and directories like file ownership, access mode, and file type the size for the older version is always fixed.

A single I-node has 13 disk addresses. The initial 10 disks will be used to point straightly to the initial 10 blocks of a file. In case of low storage (if the file is bigger than 10 blocks), the 11 address will point to the block which involves the path of the following 128 blocks of the file. In the worst-case like the size is still big enough (70,656 bytes), then the twelfth block will point at more than 128 blocks. Lastly, if the size is more than 8,459,264 bytes, the thirteenth address will be used. This will be considered the finish line of the algorithm with the maximum file size of 1,082,201,087 bytes.

A reasonable directory hierarchy has been inserted into this flat physician foundation by inserting a brand-new file to the directory. The way to have control over a directory is the same as an ordinary file. It involves 16 bytes entries that are divided by 14 bytes name and one I-number. The radicle of the hierarchy is known as I-number (viz.,2). Due to the foundation of the file system, it will allow an arbitrary, navigated graph of directories with a normal file connected at arbitrary locations in the graph. The early version of the Unix system has used this foundation. As a result, managing such a structure has become so problematic that the systems were minimized to a directory tree. All in all, the I-node is used for the inserting of directories. For achieving that, a new type of file will be added.

8. What are the primitive file system operations supported in Unix?

When it comes to the user process accesses, the user has access to the file system with certain primitives. The most well-known ones are open, create, read, write, seek, and close. There is a special location for some open files in the system information segmented linked with a user. As a result, the open file system table involves pointers which its function is noticeable in accessing corresponding I-node table entries. What has been linked to each of the open files is a present time I/O pointer. The interesting point about the read/write request is that the system will treat them randomly along with an implied search to the I/O pointer. The consideration of a file from the user perspective is sequential with the I/O pointer beyond one's control is counting the number of bytes that have been written or viewed from the file. One of the most critical tasks regarding the file-sharing is to allow the relevant process to share a common I/O pointer and for the independent processes, it should have a separate I/O pointer that can eventually approach the same file. Consequently, the I/O pointer will not be able to reside in the I-node table or occupy the list of open files for the process.

Regarding the primitives file systems, they will be executed as below.
The open operation changes the name of a path in the file system to an I-node table entry. Then the I-node's pointer will be set in a brand new formatted open file table entry.
Consequently, the file table entry's pointer will be set in the system data segment for the process.
The create operation initially makes a new I-node entry, make a note of the I-number in a directory then it will manufacture the same foundation as for an open operation. Regarding the read and write the main functionality of them is they just access the I-node entry as mentioned above. The seek operation, plays around with the I/O pointer. During the process, physical searching will not happen. Lastly, the close operation will release the structures that were manufactured by the open and create operations. However, the number of references will be saved on the open file table entry.

There is an operation called unlike which will function as decrementing the count of the numbers from the directory that are refereeing the given I-node. Also, after the last reference vanished, if no location has been left to allocate to this last one eventually it will be deleted.

Regarding the pipe, which is an unnamed FIFO type, the execution involves implied seeks before each read and write for applying FIFO.

Apart from the primitive types, the well-known Unix file types are regular, directory, symbolic link, FIFO special, block special, character special, and socket.

Reflection

First and foremost, mainly all the questions were answered by different sources but the main one that I used was the attached PDF. However, I have mentioned all the references that I used to motivate my answer. Some questions were forward, so I answered them like yes or no, then giving my reason. On the other hand, for some questions, if the reader reads the paragraphs, the question will be answered eventually. Some questions like number 3 that asked for our opinion, have been answered by giving proper explanations. I have done my best to not only answer and motivate my answer but also give more information about some questions.

Reference

- https://en.wikipedia.org/wiki/Unix
- https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming#:~:text=The%20C%20language%20was%20actually,with%20fewer%20lines%20of%20code.&text=The%20GNU%20operating%20system%20itself,components%20are%20written%20in%20C.
- https://www.geeksforgeeks.org/difference-between-operating-system-and-kernel/
- https://en.wikipedia.org/wiki/Unix
- https://en.wikipedia.org/wiki/Unix_filesystem#:~:text=File%20types,-Main%20article%3A%20Unix&text=The%20original%20Unix%20file%20system,System%20V%20added%20FIFO%20files.
- https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming
- UNIX Implementation **(***K. Thompson*, AT&T Bell Laboratories, Murray Hill, New Jersey 07974**).**
- https://en.wikipedia.org/wiki/Brain%E2%80%93computer_interface
- https://en.wikipedia.org/wiki/Solid-state_drive
- https://www.avast.com/c-ssd-vs-hdd
- https://www.avast.com/c-what-is-ssd
- https://en.wikipedia.org/wiki/Input/output
- https://www.informit.com/articles/article.aspx?p=30599
- http://shinetutorial.com/pathhlp.htm
- https://www.cs.purdue.edu/homes/bb/cs348/www-S08/unix_path.html
- https://en.wikipedia.org/wiki/Inode#:~:text=Inodes%20store%20information%20about%20files,the%20file%20system%20can%20hold.
- https://www.howtogeek.com/465350/everything-you-ever-wanted-to-know-about-inodes-on-linux/
- https://en.wikipedia.org/wiki/Unix_file_types#:~:text=The%20seven%20standard%20Unix%20file,requires%20(e.g.%20Solaris%20doors).
- Silberschatz, A., Galvin, P.B. & Gagne, G., 2014. Operating system concepts 9th ed., International student version., Hoboken, NJ: Wiley.