Linnéuniversitetet
Kalmar Växjö

Linnaeus University

1DV512 - Operating System
Individual Assignment 1
UNIX Implementation

Author: Yuyao Duan
Email: yd222br@student.lnu.se
Course Code:1DV512
Semester: Autumn 2021
Supervisors: Lars Karlsson,
Samuele Giussani, Victor Loby

Table of Contents

1. Introduction

UNIX system is recognized as one of the most powerful and popular system which is broadly used for servers, workstations, desktop and laptop [1], [2]. It enables users to perform multi-tasking and multi-user functionality [2]. There are a series of varieties of UNIX systems which are well-known in modern IT industry including Sun Solaris, Linux/GNU, and MacOS X etc. [2]. In this report, the implementation of UNIX will be investigated.

First of all, the programming language which is used for implementing UNIX kernel will be discussed. Next, a discussion based on a deeper understanding of UNIX kernel is given. Reflection on previous contributions on UNIX kernel and potential suggestions will be provided. Furthermore, the principles of running processes, I/O devices as well as the file system will be explored in details. The last but not least, the structure and operations of the file system of UNIX will be discussed respectively.

2. UNIX kernel and programming languages (Question 1-3)

UNIX system is designed with several different layers which will interact among computer hardware and the user [1]. Kernel is the first layer of the system that operates machine hardware and deal with all the connections between the hardware [1].

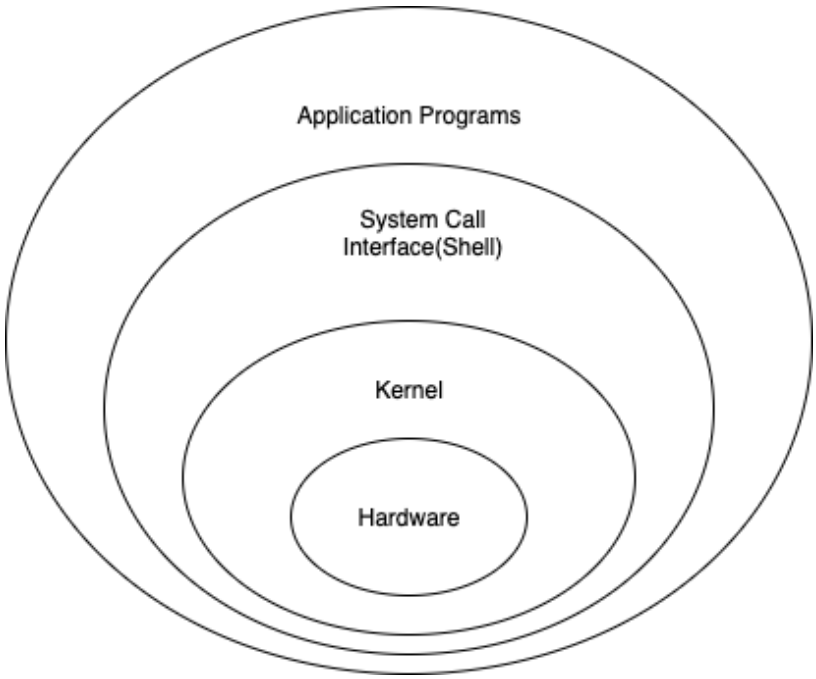

Figure 1. UNIX system architecture layer Model [1]

In order to efficiently and effectively interact with hardware, UNIX kernel consists of 10,000 lines of C code and 1,000 lines of assembly code i.e. the programming languages used for UNIX kernel are C and assembly [3]. Furthermore, in these 1000 lines of assembly code, 200 lines are implemented to improve system efficiency and 800 lines can only be programmed in assembly in order to achieve hardware functions [3].

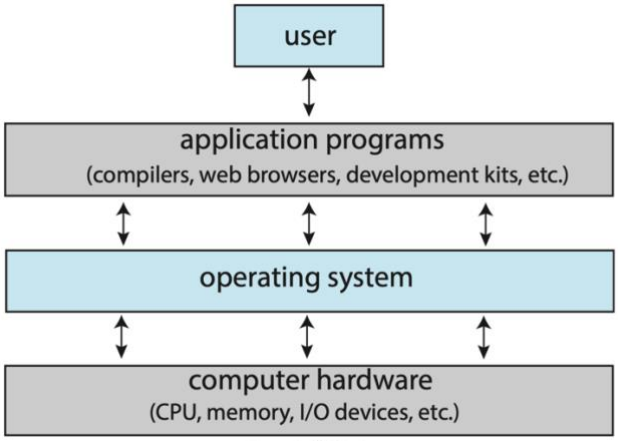

Figure 2. The role of operating system [4]

Any UNIX operating system consists of three parts: kernel, shell, and system programs [2]. Therefore, UNIX kernel is not a complete operating system. On the contrary, kernel is an important part/layer of UNIX system [2]. According to Figure 2, an operating system as a whole play the role of interacting between the computer hardware and running other application programs. As we can see from Figure 3, kernel is an important low-level layer of UNIX system which is designed to interact between hardware and system call (from shells and commands, compilers and interpreters, system libraries). The detailed functions of kernel can be identified from Figure 3.

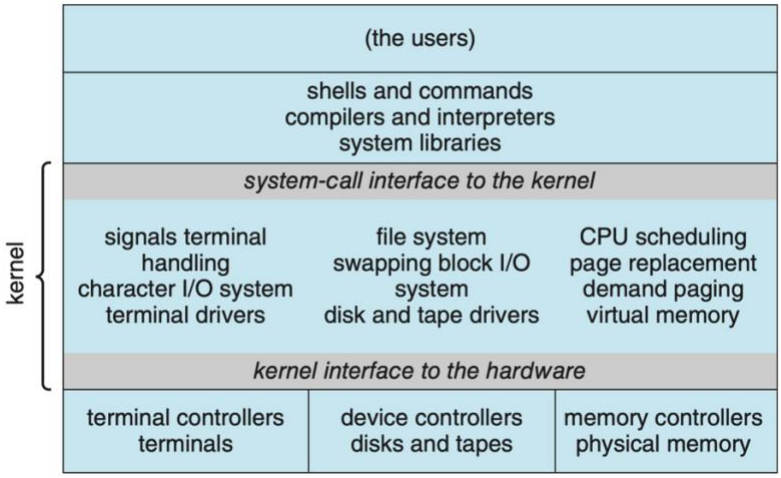

Figure 3. Kernel and UNIX operating system [4]

So far, the most commonly used languages for kernel are C and assembly language, the reason for that is because both languages are low-level programming languages which are the closest to hardware [5]. Another important reason for using C is due to this language is a memory-managed language [6]. However, compared to C, the new high-level language Rust has been considered as a competitive player for programming kernel [6]. It is a new generation language can help programmers to avoid memory leaks and buffer overflows that are very common for C users. Advantages of using Rust to program kernel include: firstly, it is high-level language that is more user-friendly for modern computer scientists; furthermore, it has automatically RAM via garbage collection similar to PHP,

Python, or Java; moreover, Rust can provide raw speed, flexibility, and direct mapping to hardware with a memory-safe environment that C is difficult to achieve [6]. Therefore, Rust is suggested as an ideal modern language to program kernel in this report.

3. UNIX system and its process control (Question 4)

In the UNIX system, the user executes programs in an environment which is called a user process [3]. When a system function is needed, the user process will call the system as a subroutine [3]. In the UNIX system, all processes are created by the system primitive fork i.e. the newly created running process (child) is a copy of original process (parent) instead of starting from scratch [3].

The relationship between the new running processes and existed running processes can be considered from the following aspects [3]. First of all, there is no noticeable sharing of primary memory between the newly created process and the old one [3]. If the parent process was running from a read-only text segments, the child will share the text segment [3]. For the writable data segments, copies will be made for the child process [3]. Files that were opened before the fork will be shared after the fork [3]. Moreover, the processes are normally allowed to select their own destiny which is due to they are informed as to their part in the relationship [3]. The parent process could wait for the termination of any child process [3]. In the case of a program wants to regain control after executing a new program, the original process should fork a child process to run the second program and the original process (parent) should wait for the child until second program finished [3].

4. UNIX I/O system (Question 5)

There are two main categories of I/O devices supported by Unix – the block I/O system (structured I/O) and character I/O system (unstructured I/O) [3]. Block I/O devices refer to the devices randomly addressed, secondary memory blocks of 512 bytes each [3]. Character I/O devices consist of all devices that do not fall into the block I/O model [3]. A block device is that its driver communicates by sending entire blocks of data including hard disks, USB cameras, Disk-On-Key etc. [7]. Based on the understanding, a Solid-State Drive (SSD) the SSD should be considered as block I/O system which is used as a secondary memory of the UNIX system [3].

A Brain-Computer Interface (BCI) is also called as neural control interface (NCI), mind–machine interface (MMI), direct neural interface (DNI), or brain–machine interface (BMI), is a direct communication path between an enhanced or wired brain and an external device [8]. According to the definition, BCI does not meet the requirement of block I/O system since it is used for researching and mapping human cognitive and sensory-motor functions [8]. The driver of BCI communicates with UNIX I/O system by sending and receiving bytes or octets [7]. Therefore, BCI is more in line with character I/O device [8].

5. UNIX file system (Question 6-8)

The file system is designed differently between UNIX system and Windows (or DOS) system. In Windows, the system normally has multiple drives like C:\, D:\, E:\ etc. However, in the UNIX system, it implements single root for file management [3]. As we can see, a file is a one-dimensional array of bytes in UNIX [3]. Files can be attached anywhere onto a hierarchy of directories [3]. A hierarchical file system has the benefit that any file or directory can be found as a child of the root directory [3]. In case, if user needs to move data to a new device, the location in the file system can stay the same which is not possible in Windows or DOS [3]. In the UNIX system, user can mount file system. In this situation, the root of this structure is the root of the UNIX file system and the second block device can be mounted at any leaf of the current hierarchy [3]. Therefore, UNIX system can attach several disk drives at the same time.

In the UNIX file system, the disk data structure is separated into four self-identifying regions: the first block (address 0), second block (address 1), i-list and free storage blocks which are used for the contents of files [3]. The third part i-list is a list of file definitions [3]. In the list, each file definition is a 64-byte structure, which is called i-node [3]. The offset of an i-node within the i-list is called its i-number [3]. I-number plays an important role in the UNIX file system since uniquely naming a particular file demands combining device name (major and minor numbers) and i-number [3]. Due to i-number is related to i-node, therefore i-node plays a significant role for naming files.

In the UNIX file system, i-nodes are used for implementation of directories. The first aspect that i-nodes contribute to directories is due to implementing a valid directory demands properly named files, thus i-nodes cannot be absent [3]. Furthermore, since the i-node defines a file, the file system needs to read i-node to access the target file. Actually, the file system needs to keep interacting with i-nodes to access files [3]. The main procedures can be generalized as following steps: first, the system locates the i-node, assign an i-node table entry, reads the i-node into primary memory; then when last access to i-node goes away, the table entry is copies back to i-list and table entry will be freed [3]. Moreover, all I/O operations need the aid of corresponding i-node table entry to access the target file [3]. A path name can be converted to an i-node table entry swiftly, and by utilizing i-node the system can easily search and read the target directory [3].

UNIX supports the following primitive file system operations: open, create, read, write, seek, close, and unlink, the detailed descriptions see Table 1.

Table 1. The supported primitive file system operations in the UNIX system [3]

Primitive file system operations	Description
Open	Transforming the path name to an i-node table entry which will be used as mentioned above
Create	Creating a new i-node entry and write the i-number to a directory

Read	Accessing target i-node entry
Write	Accessing target i-node entry
Seek	Operating the I/O pointer instead of physical seeking
Close	Free the target i-node table entry
Unlink	Reducing the count of the number of directories pointing at the given i-node

6. Reflection

This assignment provides a valuable chance to systematically study the structure and the underlying principles of the UNIX system. The system consists of three main parts: kernel, shell, and system programs. Based on C and assembly languages, UNIX system can efficiently manipulate hardware. We can also see that the state-of-art high-level programming language such as Rust could also add potential for programming the kernel in terms of efficiency and user-friendly experience. From the report we could learn that UNIX system applies an efficient process control strategy that the new running process is a copy of parent process which greatly improves the efficiency of processes initialization. UNIX system categorizes I/O devices in two ways: block I/O system and character I/O system that could cover all the requirements of devices. Unlike Windows or DOS system, UNIX system implements single root for file management that files are attached onto hierarchical directories. This design does contribute UNIX-like systems such as Linux, Solaris, FreeBSD become more suitable candidates for web servers and enterprise applications.

References

[1] Ukessays, "*Importance Of Unix Operating System Information Technology Essay*", [2021-11-10], url: [https://www.ukessays.com/essays/information-technology/importance-of-unix-operating-system-information-technology-essay.php]

[2] Educba, "*Uses of Unix*", [2021-11-10], url: [https://www.educba.com/uses-of-unix/]

[3] Thompson, K., "*UNIX Implementation*", [2021-11-10], url: [https://users.soe.ucsc.edu/~sbrandt/221/Papers/History/thompson-bstj78.pdf]

[4] Silberschatz A, Galvin P, Gagne G, Operating Systems Concepts (Tenth Edition), International Student Version, John Wiley & Sons, ISBN 978-1-118-06333-0.

[5] Developers, "*After All These Years, the World is Still Powered by C Programming*", [2021-11-10], url: [https://www.toptal.com/c/after-all-these-years-the-world-is-still-powered-by-c-programming]

[6] Arstechnica, "*Linus Torvalds weighs in on Rust language in the Linux kernel* ", [2021-11-10], url: [https://arstechnica.com/gadgets/2021/03/linus-torvalds-weighs-in-on-rust-language-in-the-linux-kernel/]

[7] Tutorialspoint, "*Operating System - I/O Hardware*", [2021-11-13], url: [https://www.tutorialspoint.com/operating_system/os_io_hardware.htm]

[8] Krucoff MO, Rahimpour S, Slutzky MW, Edgerton VR, Turner DA, "*Enhancing Nervous System Recovery through Neurobiologics, Neural Interface Training, and Neu-rorehabilitation*", [2021-11-10], url: [https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5186786/]

Appendix: Questions for report research

Tasks

Please provide (and motivate) answers for the following questions (all questions refer to the Unix version discussed in Thompson's report):

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

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

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?

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

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)?

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?

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

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