FreeBSD

ITP51 – OPERATING SYSTEMS

TABLE OF CONTENTS

CHAPTER 1

Introduction and History

During the 1990s, FreeBSD materialized from the aspirations of a group of talented individuals. Their vision aimed to shape a world where software was not just free but also infused with expertise and commitment. FreeBSD originated as an open-source operating system (OS) derived from the extensive Berkeley Software Distribution (BSD) Unix. Evolving with technological advancements, FreeBSD has gained prominence in the realms of security, reliability, and its capacity to support intricate server services. Operating under an open-source framework, FreeBSD provides developers the flexibility to customize, striving to offer meticulous control over every facet of the system. These dreams weren't mere utopian ideals; they were aspirations with identifiable individuals who navigated the technological landscape, breathing life into the concept of a free operating system. Amid challenges and learning experiences, FreeBSD emerged as a triumphant project. This vision was shaped by individuals dedicated to the principles of free software, resulting in a system that not only embraced the path of freedom but also delivered top-notch security and performance. From a technical standpoint, the FreeBSD kernel establishes a robust foundation for a swift and dependable system, accommodating a spectrum of applications from basic web servers to expansive network infrastructure. FreeBSD's reliability underscores the notion that free software can be a driving force in technology. Notably, FreeBSD stands out for its customization capabilities, liberating users from a rigid structure and empowering sysadmins and developers to unleash their creativity and realize their ideas. Beyond the lines of code and technical concepts, the success of FreeBSD is intricately tied to documentation and community collaboration. Over time, this

project has served as a conduit for those who contribute, teach, and learn, fostering collective excellence that imbues the spirit of free software, where everyone shares in a common goal. FreeBSD transcends its identity as a mere operating system; it embodies a narrative of success and unity. With each new day, it persists as an inspiration for those striving to transform the landscape of technology. Every idea finds a place, and every contributor becomes part of a broader journey in the realm of free software. Thanks to FreeBSD, the dream of a free operating system has not only materialized but also continues to unfold as an ongoing adventure, continually infusing meaning into the endeavors of those seeking change and collaboration.

CHAPTER 2

Process Management

In advanced process management system, FreeBSD emerges as a leading operating system that not only provides high performance but also offers excellent control and security in every aspect of the system process. In the digital world, effective process management is a fundamental aspect of any operating system's operation, and in the case of FreeBSD, it is an aspect that goes beyond efficiency and reliability. It has mechanisms such as inter-process communication (IPC) and synchronization primitives that allow processes to interact and communicate with each other seamlessly. Proper coordination of processes enables them to share information and collaborate for the overall performance of the system FreeBSD, an open-source operating system originating from BSD Unix, is known for its advanced process management mechanisms aimed at enhancing the performance, capabilities, and efficiency of the entire system. One of the key components of process management in FreeBSD is its

kernel scheduler. The kernel scheduler is the part of the operating system that determines which process should be given access to the CPU and how it should be done. In FreeBSD, the kernel scheduler is recognized for its effective algorithm, resulting in a smooth distribution of CPU time among different processes. This allows for higher overall system performance, especially in environments that require the swift execution of tasks. Additionally, FreeBSD provides support for advanced features in process management, such as threading. Threading is a method of performing concurrent or parallel tasks within a process. FreeBSD's advanced thread management enables the efficient utilization of multicore processors, bringing higher levels of performance and responsiveness to modern hardware. The advancement of technology also brings new challenges to security, and this is where mechanisms like Jail in FreeBSD come into play. Jail is a feature that provides process-level virtualization, offering separate space and environments for each process. This contributes to security and isolation, particularly in server environments that require protection against potential attacks and the creation of weaponized containers. With its excellent implementation of advanced process management mechanisms, FreeBSD reflects excellence and the ability

to keep pace with the modern needs of users and developers. Careful examination of the performance of each system component, coupled with a focus on development and security, unveils the narrative of FreeBSD's success in the field of process management. The harmonious and effective integration of technology, efficiency, and reliability bestows upon FreeBSD that leads in the world of operating systems.

CHAPTER 3

CPU Scheduling

In the FreeBSD kernel, CPU scheduling plays a crucial role in managing processes, ensuring the equitable distribution of CPU time among different tasks. A comprehensive grasp of CPU scheduling is essential for understanding the operating system's performance, particularly FreeBSD, and its successful adaptation to the requirements of modern computing environments. CPU scheduling involves determining which process should have access to the CPU and for how long. Ultimately, the scheduler makes decisions that dictate the sequence of processes and allocates CPU time to each one. The FreeBSD Scheduler stands out for its efficiency in CPU time allocation, serving as a critical component operating under the time-sharing system. Employing the Unix-Like Environment (ULE) Scheduler algorithm, it is a dynamic priority scheduler based on the principles of complete CPU time and weighted response time. Recognized as an advanced mechanism, the ULE Scheduler can adapt to the diverse needs of different processes. Emphasizing fairness and responsiveness, the ULE Scheduler ensures a fair distribution of CPU time among various processes, promoting equal opportunities for all. Responsiveness focuses on the system's quick response to tasks and proper CPU time allocation, crucial for multithreaded applications and modern computing with an emphasis on real-time activities. A pivotal element of the ULE Scheduler is the Run Queue, a list of processes waiting to be processed or scheduled on the CPU. The ULE Scheduler adeptly manages and ensures the proper arrangement of CPU time. Processes with higher priority or lower weighted response time are more likely to secure CPU time. In Load Balancing, the ULE Scheduler strives to distribute tasks among different CPU cores in a multicore system, optimizing the use of multicore processors and enhancing overall performance. While the ULE Scheduler is known for its meticulous design, FreeBSD also provides various tunable parameters that can be adjusted by the system administrator. This

flexibility allows customization of the scheduler to specific needs or environments, providing control over its behavior, including settings for time quantum, thread priorities, and other critical scheduling aspects. Preemption is another integral aspect of CPU scheduling in FreeBSD, enabling a higher-priority process to preempt the CPU from a lower-priority process. Overall, CPU scheduling in FreeBSD proves to be an effective mechanism, incorporating advanced features and algorithms to deliver high performance and responsiveness across diverse computing environments.

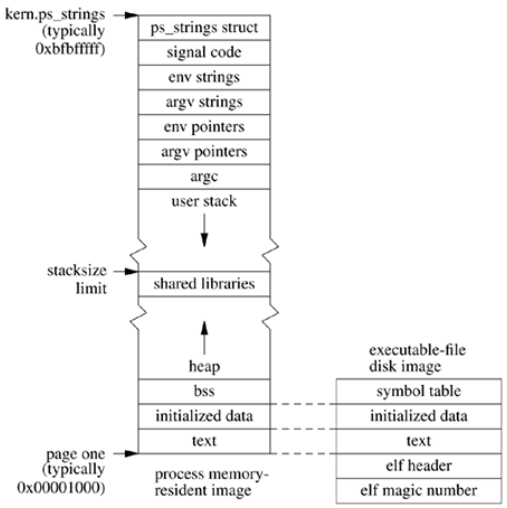
CHAPTER 4

Memory Management

Memory Management in FreeBSD is a fundamental part of the kernel that aims to provide effective allocation and utilization of memory resources throughout the system. A thorough examination of FreeBSD's Memory Management opens the door to understanding how it succeeds in overseeing and addressing memory-related tasks in modern computing environments. The Virtual Memory System of FreeBSD is an advanced mechanism that ensures the allocation and usage of memory for each process in the system. Each process is given a virtual address space containing code, data, and other parts of the program. This virtual address space is not directly tied to real memory but is managed by the FreeBSD Memory Manager to cater to the needs of each process. The Page System is a crucial part of Memory Management. Memory is divided into small units called "pages" with a fixed size. The Page System allows dynamic allocation and utilization of memory, providing meticulous control over each memory unit. Unnecessary pages can be moved to secondary storage or returned to the free pool for other processes to use. The Swap System provides a mechanism for virtual memory paging. When the system's physical memory is full, inactive pages are transferred to a swap space on the hard disk. This offload inactive parts of memory to make

room for other processes to use. The Swap System enables virtual memory to interact with a larger secondary storage. Memory Protection is also a significant part of Memory Management in FreeBSD. Memory protection is implemented to prevent one process from accessing or writing to the memory of another process. It allows each process to have its own memory space that cannot directly interfere with other processes, ensuring security and isolation between them. The Copy-On-Write (COW) Mechanism provides an effective way to handle

memory during fork operations, a crucial part of the process of creating a new process. Instead of creating an exact copy of a process's memory, the COW Mechanism promotes the use of the same memory space between the parent and child processes. When a write occurs in memory, only then does the memory of the child process become independent. The Kernel Address Space is a critical aspect of Memory Management. The FreeBSD kernel has its own address space hidden from user processes. This provides a separate space for kernel data structures and code. Hiding the kernel address space from user processes protects the kernel against unauthorized access and enhances the overall system's security. Memory Mapping provides the ability to associate a file with a virtual address space. It allows user processes to directly access a file using memory operations. Memory Mapping enables the efficient performance of operations such as reading and writing large datasets without needing to load the entire file into memory.



CHAPTER 5

Storage Management

Storage Management in FreeBSD is a critical part of the kernel and operating system that enables effective allocation, access, and maintenance of storage resources. A detailed examination of FreeBSD's Storage Management provides insights into the mechanisms and principles that contribute to the success of this operating system in meeting the needs of modern computing environments. File Systems are a fundamental component of Storage Management. FreeBSD supports various file systems such as UFS (Unix File System), ZFS (Zettabyte File System), and FAT (File Allocation Table). UFS is a traditional file system known for its simple design and stability. ZFS is an advanced file system featuring beneficial features like data integrity, snapshotting, and RAID-Z. FAT, or File Allocation Table, is a simple file system often used in various storage devices. Disk Management enables effective utilization and access to storage devices such as hard drives and solid-state drives. FreeBSD employs mechanisms like GEOM (Generic Encapsulation of Disks Mechanism) that provide an abstraction layer for different types of storage devices. This allows the operating system to interact with storage devices without directly interfering with their specific configurations. RAID Support allows FreeBSD to combine multiple storage devices to enhance performance, reliability, or both. RAID (Redundant Array of Independent Disks) provides disk mirroring, striping, parity, and other techniques to improve data storage performance and reliability.

The partition scheme is created, and then a single partition is added. To improve performance on newer disks with larger hardware block sizes, the partition is aligned to one-megabyte boundaries

```
# gpart create -s GPT ada1
# gpart add -t freebsd-ufs -a 1M ada1
```

Depending on use, several smaller partitions may be desired. See gpart for options to create partitions smaller than a whole disk.

The disk partition information can be viewed with gpart show

```
% gpart show ada1
=>        34  1465146988  ada1  GPT  (699G)
          34        2014        - free -  (1.0M)
        2048  1465143296     1  freebsd-ufs  (699G)
  1465145344        1678        - free -  (839K)
```

CHAPTER 6

I/O Systems

The Input/output (I/O) Systems in FreeBSD are an integral part of the kernel responsible for the distribution and management of data between peripheral devices and memory. A deep understanding of FreeBSD's I/O Systems reveals mechanisms and structures that enable the effective use of input and output resources, showcasing the capabilities of this operating system in the fields of storage, networking, and other peripherals. **Device Drivers** are a fundamental component of the I/O Systems in FreeBSD. These are software modules that facilitate communication between the kernel and hardware devices such as network interfaces, storage controllers, and others. **Broad support** for various device drivers allows FreeBSD to be compatible with a wide range of hardware, providing flexibility for users and administrators. **Buffer Cache** is a mechanism that allows the temporary storage of data in memory before it is passed to or retrieved from a device. Using the buffer cache results in faster data access because operations can be performed in memory without having to wait for the actual storage device. It is an optimization for I/O operation performance. **I/O Schedulers** provide a mechanism for selecting the sequence of I/O operations to be performed on a

storage device. **The ULE Scheduler**, known for CPU scheduling, exhibits similar behavior in adjusting the sequence of I/O operations based on its dynamic algorithm. Using I/O schedulers leads to more effective utilization of storage devices and enhances overall system performance. **Direct Memory Access (DMA)** is a mechanism that allows peripheral devices to access memory without involving the CPU. It enables faster and more efficient data transfer between devices and memory. DMA is a crucial mechanism that speeds up I/O operations and enhances overall system performance. **I/O Request Queue** allows the system to effectively manage how I/O requests from various processes are processed. Queue management provides an opportunity for I/O requests to wait appropriately before being passed to their target storage device. It is part of disk scheduling and provides a mechanism for adjusting the sequence of I/O requests. Interrupt Handling is a critical part of the I/O Systems in FreeBSD. An interrupt is a signal sent by a hardware device to the CPU to notify that attention is needed. Interrupt handling enables the kernel to respond promptly to asynchronous events from peripherals, such as after a data transfer or completion of an I/O operation. **File Descriptors** enable processes to hold references to open files and devices. It provides abstraction for processes to interact with various resources without directly interacting with them. The use of file descriptors allows processes to be more modular and not tied to a specific device or file.

CHAPTER 7

File Systems

The File Systems of FreeBSD are an integral part of the kernel and operating system that provide structure and management for storing, accessing, and handling data on storage

devices. Through a thorough examination of FreeBSD's File Systems, we gain a deeper understanding of the mechanisms and principles that bring the data execution system of this operating system to life. The Unix File System (UFS) is a traditional file system known for its simple design and robust performance. It utilizes inode to support file metadata, including file permissions, ownership, and timestamps. UFS is capable of accommodating long file names, symbolic links, and protecting files and directories against accidental deletion. The Zettabyte File System (ZFS) is an advanced file system featuring beneficial features such as data integrity, copy-on-write, and automatic snapshotting. It is a modern file system offering high-level resilience and performance. ZFS supports dynamic disk striping, mirroring, and RAID-Z, providing options for redundancy and data loss recovery. The File Allocation Table (FAT) is a simple file system commonly used in various storage devices such as USB drives and memory cards. It comes in versions like FAT12, FAT16, and FAT32. FAT supports a simple design, making it easy to support and understand across different systems, albeit with limited features compared to other more modern file systems. ISO 9660 is a standard file system for optical disc storage such as CD and DVD. It includes directory structures, file attributes, and file names. ISO 9660 has restrictions on file name length and file path, but it is considered a standard enabling interoperability across different operating systems. Network File System (NFS) is a distributed file system protocol allowing computers to access files on different computers like local files. NFS supports sharing and accessing files on various networked systems, providing transparency between the client and server for seamless use of remote files. MSDOS File System (MSDOSFS) is a file system tailored for MS-DOS and Windows, enabling interoperability between FreeBSD and storage devices formatted using MS-DOS or FAT. It supports FAT12, FAT16, and FAT32, known for its simple and straightforward design. Soft Updates is an advanced feature of the file system in FreeBSD aiming to improve the reliable performance of file systems even during system crashes or power failures. Instead of using traditional journaling, Soft Updates employs a dependency system to maintain file

system consistency despite changes. Journaling is a file system mechanism allowing quick recovery from system crashes. Instead of recording each change to the file system, they are logged in a journal, and the file system can easily recover from the last journal entry. Journaling allows faster reboots and quicker recovery from accidents. Union Mount enables multiple file systems to be maintained and merged simultaneously.