VILNIUS GEDIMINAS TECHNICAL UNIVERSITY

FACULTY OF ELECTRONICS

DEPARTMENT OF COMPUTER SCIENCE AND COMMUNICATIONS TECHNOLOGIES

STRUCTURE OF MODERN OPERATING SYSTEMS: LINUX

Modern Operating Systems

Abstract

Author: Ozan Berk Gultegin (DISfmu-24)

Lecturer: Prof. Dr. (HP) Vytautas Urbanavičius

VILNIUS 2025

Table of Contents

STRUCTURE OF MODERN OPERATING SYSMS: LINUX 1
1. INTRODUCTION 3
1.1. Stages of Development of Linux: 3
2. LINUX COMPONENTS 4
3. SERVICES PROVIDED BY LINUX 5
4. SYSTEM PROGRAMS 6
5. PROCESS MANAGEMENT 7
6. CPU SCHEDULING 8
7. PROCESS SYNCHRONIZATION 9
8. MEMORY MANAGEMENT 10
9. VIRTUAL MEMORY 11
10. FILE SYSTEM 12
11. I/O SYSTEM 13
12. SECURITY FEATURES 14
13. CONCLUSION 15
14. REFERENCES 15

1. INTRODUCTION

Linux is an open-source, UNIX-like operating system kernel initially created by Linus Torvalds in 1991. Today, it forms the foundation for numerous operating system distributions used across devices ranging from smartphones and embedded systems to personal computers and the world's most powerful supercomputers. This abstract examines the structure of Linux as a modern operating system, exploring its key components, services, management mechanisms, and architectural design principles that have contributed to its widespread adoption and remarkable versatility.

1.1. Stages of Development of Linux:

- Origins (1991-1994):

Linux began as a personal project by Finnish computer science student Linus Torvalds, who sought to create a free operating system kernel for his Intel 386-based computer. The first version (0.01) was released in September 1991 with approximately 10,000 lines of code. In its earliest form, Linux could run bash (Bourne Again SHell) and gcc (GNU C Compiler) but had limited hardware support.

- Early Growth (1994-2000):

The adoption of the GNU General Public License v2 encouraged collaborative development. Version 1.0 was released in 1994, marking the first production-ready kernel. During this period, Linux gained essential features like symmetric multiprocessing support, improved networking capabilities, and broader hardware compatibility.

- Maturation (2001-2011):

Linux 2.4 (2001) introduced major advancements including USB support, ISA Plug-and-Play, and improved SMP scaling. Version 2.6 (2003) brought significant improvements to the scheduler, memory management, and file systems. This era saw Linux expand from servers into embedded systems and mobile devices.

- Modern Era (2011-Present):

Version 3.0 was released in 2011, followed by version 4.0 in 2015 and version 5.0 in 2019. Recent development has focused on improving container support, real-time capabilities, security enhancements, and support for emerging hardware architectures. The development process has matured with a time-based release cycle, delivering new stable versions approximately every 9-10 weeks.

2. LINUX COMPONENTS

The Linux operating system consists of several key components that work together:

Kernel

The kernel is the core component that manages system resources and acts as an interface between hardware and software. It handles memory allocation, process scheduling, device drivers, and system calls.

System Libraries

These provide standard functions that applications can use to access kernel features. The most important is the GNU C Library (glibc), which provides the system call interface and basic functions specified by the POSIX standard.

System Utilities

These are programs that perform specialized individual management tasks. Examples include utilities for configuring network interfaces (ifconfig, ip), managing processes (ps, top), and maintaining file systems (fsck).

Shell

The shell provides the user with an interface to the operating system. Popular Linux shells include Bash (Bourne Again SHell), Zsh, and Fish. The shell interprets user commands and provides mechanisms for scripting.

Windowing System

The X Window System (X11) or Wayland provides the standard toolkit and protocol for building graphical user interfaces and managing display.

3. SERVICES PROVIDED BY LINUX

Linux offers a wide range of services to both users and applications.

- **Program Execution**

Linux loads programs into memory and ensures their proper execution by allocating CPU time and handling their system calls.

- **I/O Operations**

The kernel manages input/output operations to and from hardware devices, providing abstraction layers that simplify device access for applications.

File System Manipulation

Linux provides a unified interface for file operations (create, delete, read, write) across various storage devices and file system types.

- **Communications**

Linux supports interprocess communication through mechanisms like sockets, pipes, and shared memory, as well as network communications through comprehensive protocol implementations.

- **Error Detection**

The kernel detects and handles hardware errors, memory protection violations, and other system faults, preventing individual application failures from crashing the entire system.

- **Resource Allocation**

Linux tracks resource usage and allocates CPU time, memory, and I/O devices to processes based on scheduling and prioritization algorithms.

- **Security and Protection**

The kernel enforces access controls, user permissions, process isolation, and other security mechanisms to protect system integrity and user data.

4. SYSTEM PROGRAMS

Linux distributions include various system programs that provide functionality beyond the basic kernel services:

System Utilities

- **Package Managers**: dpkg/apt (Debian/Ubuntu), rpm/dnf/yum (Red Hat/Fedora), pacman (Arch)
- **System Monitors**: top, htop, glances
- **User Management**: useradd, passwd, sudo
- **Network Utilities**: ping, traceroute, ip, ss

Development Tools

- **Compilers**: GCC (GNU Compiler Collection), Clang
- **Build Systems**: make, cmake, meson
- **Debuggers**: gdb, strace, ltrace
- **Version Control**: git, subversion

Graphical Interface Components

- **Display Servers**: X.Org Server, Wayland compositors
- **Desktop Environments**: GNOME, KDE Plasma, Xfce
- **Window Managers**: i3, Openbox, Awesome

System Daemons

- **systemd**: Init system and service manager
- **cron**: Time-based job scheduler
- **sshd**: Secure Shell daemon
- **NetworkManager**: Network configuration daemon

5. PROCESS MANAGEMENT

Process Representation

Linux represents processes using task_struct structures in the kernel, which contain all information about a process, including its state, priority, memory usage, open files, and signal handlers.

Process Creation

New processes are created through the fork() system call, which creates a copy of the calling process. The exec() family of system calls is then typically used to replace the process image with a new program.

Process States:

Linux processes can be in various states:

- TASK_RUNNING: Process is either executing or waiting to be executed
- TASK_INTERRUPTIBLE: Process is sleeping until a condition becomes true
- TASK_UNINTERRUPTIBLE: Process is sleeping but can't be interrupted by signals
- TASK_STOPPED: Process execution has been stopped
- TASK_ZOMBIE: Process has terminated but parent hasn't read its exit status

Threads in Linux

Linux implements threads using the same mechanisms as processes. From the kernel's perspective, threads are simply processes that share certain resources. The clone() system call creates threads with specified shared resources.

6. CPU SCHEDULING

Scheduler Classes:

Linux uses a modular scheduler architecture with multiple scheduler classes:

- Completely Fair Scheduler (CFS): Handles normal processes
- Real-Time Scheduler: For real-time processes with strict timing requirements
- Deadline Scheduler: For processes with specific completion deadlines
- Idle Scheduler: Runs when there's nothing else to execute

Completely Fair Scheduler (CFS)

The CFS aims to give each process a fair share of CPU time based on its weight (priority). It uses a red-black tree to track runnable processes and selects the one with the lowest "virtual runtime" to execute next. This ensures that processes receive CPU time proportional to their priority while maintaining good interactive performance.

Scheduling Policies

Linux supports several scheduling policies:

- SCHED_OTHER (normal): Default policy for regular processes
- SCHED_FIFO: First-in, first-out real-time policy
- SCHED_RR: Round-robin real-time policy
- SCHED_BATCH: For CPU-intensive batch processes
- SCHED_IDLE: For extremely low-priority tasks
- SCHED_DEADLINE: For processes with specific deadlines

Load Balancing

On multiprocessor systems, Linux performs periodic load balancing to distribute processes across CPUs. It considers factors like cache locality, CPU topology, and process affinity to optimize performance.

7. PROCESS SYNCHRONIZATION

- **Synchronization Mechanisms:**
  Linux provides several synchronization primitives

- **Atomic Operations:**

Basic operations that execute without interruption, forming building blocks for higher-level synchronization constructs.

- **Spinlocks:**

Simple locks where a process continuously checks if a lock is available. Used for short-duration critical sections where sleeping would be inefficient.

- **Semaphores:**

Counting semaphores allow a limited number of processes to access a resource concurrently. Binary semaphores (mutexes) allow only one process at a time.

- **Read-Write Locks:**

Allow concurrent access for multiple readers but exclusive access for writers.

- **RCU (Read-Copy-Update):**

A synchronization mechanism designed for read-mostly situations, allowing readers to proceed without locking while ensuring safe updates.

- **Futexes (Fast User-space Mutexes):**

Futexes provide efficient synchronization by combining user-space operations with kernel support. They avoid unnecessary system calls when there's no contention for a lock.

- **Prevention of Priority Inversion:**

Linux employs priority inheritance to prevent priority inversion problems, where a high-priority process waits for a resource held by a low-priority process.

8. MEMORY MANAGEMENT

Memory Layout:

Linux divides physical memory into three zones:

- ZONE_DMA: For DMA-capable devices with addressing limitations
- ZONE_NORMAL: For regularly mapped memory
- ZONE_HIGHMEM: For memory beyond the kernel's direct mapping range

Page Frame Management:

The kernel tracks physical memory using page frames (typically 4KB blocks). The buddy allocator manages physical pages, grouping them in powers of two to minimize fragmentation.

Slab Allocator:

Built on top of the buddy system, the slab allocator manages kernel object caching and allocation. It reduces fragmentation and improves performance by recycling objects of the same type.

User Space Memory Allocation:

User space memory is managed through virtual memory areas (VMAs). The kernel tracks memory regions using a red-black tree of vm_area_struct objects, representing memory segments with different permissions and properties.

Memory Reclamation:

Linux uses a page replacement algorithm called the LRU (Least Recently Used) list to decide which pages to evict when memory is low. Pages are categorized as either active or inactive, with the least recently used inactive pages being prime candidates for reclamation.

9. VIRTUAL MEMORY

Address Spaces:

Linux provides each process with a virtual address space of up to 4GB (32-bit) or 128TB (64-bit), isolated from other processes. This creates the illusion of having access to more memory than physically available.

Page Tables:

Linux uses a multi-level page table structure to translate virtual addresses to physical addresses:

- 32-bit systems typically use a 2-level page table
- 64-bit systems typically use a 4-level page table (with 5-level support in newer kernels)

Demand Paging:

Pages are loaded into physical memory only when accessed, not when allocated. This allows processes to use virtual memory spaces larger than physical memory.

Memory-Mapped Files:

Linux supports mapping files directly into memory, allowing file I/O to be performed through memory operations rather than explicit read/write system calls.

Swap Space:

When physical memory is exhausted, Linux can move less frequently used pages to disk-based swap space, freeing physical memory for active processes.

Transparent Huge Pages (THP):

Linux supports using larger page sizes (typically 2MB instead of 4KB) to reduce TLB (Translation Lookaside Buffer) misses and improve performance for applications using large amounts of memory.

10.FILE SYSTEM

Virtual File System (VFS):

The VFS provides a unified interface for multiple file system types. It abstracts common file operations and allows transparent access to different file systems and storage devices.

Common File Systems:

- **Ext4**

The fourth extended file system is the default in many distributions. It supports large file sizes (up to 16TB), journaling for reliability, and extent-based storage for handling of large files.

- **XFS**

Designed for scalability, XFS excels with large files and high-performance environments. It supports file sizes up to 8EB and volumes up to 16EB.

- **Btrfs**

A modern copy-on-write file system with advanced features including snapshots, compression, and RAID-like functionality.

- **F2FS**

The Flash-Friendly File System is optimized for flash-based storage devices like SSDs and eMMC.

- **File System Operations**

Linux file systems implement basic operations including:

- Creating and removing files and directories
- Opening, reading, writing, and seeking within files
- Managing permissions and ownership
- Setting and retrieving file attributes

Journaling:

Many Linux file systems use journaling to maintain integrity during system crashes. Before making changes to file system structures, operations are recorded in a journal, allowing recovery after unexpected shutdowns.

Buffer Cache:

Linux maintains a page cache that stores recently accessed file data in memory. This improves performance by reducing disk access and enabling read-ahead optimizations.

Inodes

Files are represented internally using inodes, which store metadata including permissions, ownership, timestamps, and pointers to data blocks. Each file system has its own inode structure, abstracted through the VFS.

11. I/O SYSTEM

Device Drivers

Linux includes thousands of device drivers that provide interfaces between hardware devices and the kernel. These are organized into categories like block devices, character devices, and network devices.

I/O Scheduling

The I/O scheduler optimizes disk access patterns to improve throughput and reduce latency:

- CFQ (Completely Fair Queuing): Provides fair access to disk resources
- Deadline: Ensures no request is starved of service
- NOOP: Minimal scheduling for devices with their own optimizations (like SSDs)
- MQ-Deadline: Multi-queue version of Deadline for modern storage devices

Block Layer

The block layer manages block devices (like disks) and provides services like request queuing, merging, and sorting. It interfaces between file systems and physical storage devices.

Device Files

Linux represents devices as special files in the /dev directory. These include character devices (accessed sequentially) and block devices (random access).

12.SECURITY FEATURES

Discretionary Access Control (DAC):

The traditional UNIX permission model controls access based on user/group ownership and permission bits (read, write, execute) for owner, group, and others.

Mandatory Access Control (MAC):

Linux supports MAC frameworks that enforce system-wide security policies:

- SELinux (Security-Enhanced Linux): Implements fine-grained access controls based on security contexts
- AppArmor: Provides path-based access controls using application profiles

Capabilities:

Linux divides root privileges into distinct capabilities that can be granted individually, allowing for principle of least privilege implementation.

Secure Boot:

Linux supports secure boot mechanisms to verify the integrity of the boot process and prevent unauthorized kernel loading.

Namespaces:

Namespaces isolate system resources (processes, network, users, etc.) between different groups of processes, forming the foundation for container technologies like Docker.

Control Groups (cgroups):

Cgroups allow limiting, accounting, and isolating resource usage (CPU, memory, disk I/O) of process groups.

13.CONCLUSION

Linux represents a sophisticated and versatile modern operating system that has evolved significantly from its modest beginnings. Its layered structure, with the kernel at the core surrounded by system libraries, utilities, and user applications, provides a robust and flexible foundation for diverse computing environments. The combination of process management capabilities, advanced memory management features, modular file system architecture, and comprehensive security mechanisms enables Linux to excel in roles ranging from embedded devices to enterprise servers.

The open-source nature of Linux has been instrumental in driving its rapid evolution and adaptation to new technologies and requirements. As computing continues to evolve, Linux's modular design and robust resource management capabilities position it to remain at the forefront of operating system technology, adapting to emerging hardware architectures, virtualization technologies, and security challenges.

14.REFERENCES

1- Love, R. (2010). *Linux Kernel Development* (3rd ed.). Addison-Wesley Professional.

2- Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux Kernel* (3rd ed.). O'Reilly Media.

3- Corbet, J., Rubini, A., & Kroah-Hartman, G. (2005). *Linux Device Drivers* (3rd ed.). O'Reilly Media.

4- Kerrisk, M. (2010). *The Linux Programming Interface*. No Starch Press.

5- Mauerer, W. (2008). *Professional Linux Kernel Architecture*. Wrox.

6- The Linux Kernel documentation. https://www.kernel.org/doc/html/latest/

7- Linux man-pages project. https://www.kernel.org/doc/man-pages/