Sasken Summer Internship Program - 2025(6th Sem): Linux-C stream

Project Description:

Develop a user-space tool that periodically reads /proc/slabinfo and /proc/vmstat to identify growing slab caches or suspiciously increasing kernel memory allocations. Analyze memory usage over time, detect objects that never get freed, and generate alerts when usage crosses defined thresholds. Provide a CLI to list top-N slab caches by growth rate and visualize trends in text or simple charts.

Nethra K	nethrak05052004@gmail.com
Nikitha Ullas	nikithaullas40@gmail.com
Preethi M	preethi.naik04@gmail.com
Rajanish Shetty	rajanishshetty1009@gmail.com

CONTENTS

Chapter 1

INTRODUCTION

In the intricate world of modern operating systems, the kernel stands as the central orchestrator, meticulously managing system resources to ensure smooth and efficient operation. Among its many critical functions, memory management holds paramount importance, particularly in operating systems like Linux. Kernel components constantly engage in the allocation and deallocation of memory for various data structures, relying on specialized memory allocators such as the slab allocator. This dynamic memory handling is fundamental to the kernel's ability to support diverse processes and operations.

However, a significant challenge arises when these meticulously managed memory allocations are not properly freed. This can occur due to defects in poorly written device drivers or kernel modules, leading to what is known as a memory leak. Unlike memory leaks in user-space applications, kernel memory leaks pose a far more severe threat to system stability and performance. A user-space leak might cause a single application to crash, but a kernel memory leak can directly compromise the entire operating system, potentially leading to system crashes, degraded overall performance, or even complete system freezes. The insidious nature of kernel memory leaks makes their detection and resolution a complex and sophisticated challenge, especially when compared to the more readily available profiling tools for user-space programs.

Given the critical impact of kernel memory leaks, there is a clear need for robust tools to identify and address them. This project directly confronts this challenge by developing a C-based user-space tool. The primary aim is to provide a practical solution for detecting kernel memory leaks by monitoring and analyzing slab cache behavior over time. The tool achieves this by reading vital memory statistics from the

/proc/slabinfo and /proc/vmstat virtual filesystems. By analyzing the data extracted from these files, the tool can identify suspicious growth patterns in slab cache usage. When such patterns are detected, the tool is designed to alert users through a straightforward command-line interface (CLI) that includes a textual graphical view for better visualization. This approach allows for proactive identification of potential kernel memory issues, thereby contributing to enhanced system reliability and performance.

Chapter 2

Objective

The objective of this project is to design and implement a lightweight, modular, user-space tool using the C programming language, which can:

- Periodically read kernel memory usage statistics from /proc/slabinfo
- Log slab cache details (cache name, active objects, total objects)
- Analyze log data to detect abnormal growth trends
- Display results using a command-line interface
- Generate alerts when growth exceeds defined thresholds
- Visualize slab cache growth using ASCII bar graphs
- Implement basic testing for each module
- Demonstrate OS concepts such as file I/O, process management, CLI interaction, and testing

Chapter 3

System Architecture and Workflow

3.1 Architecture Components

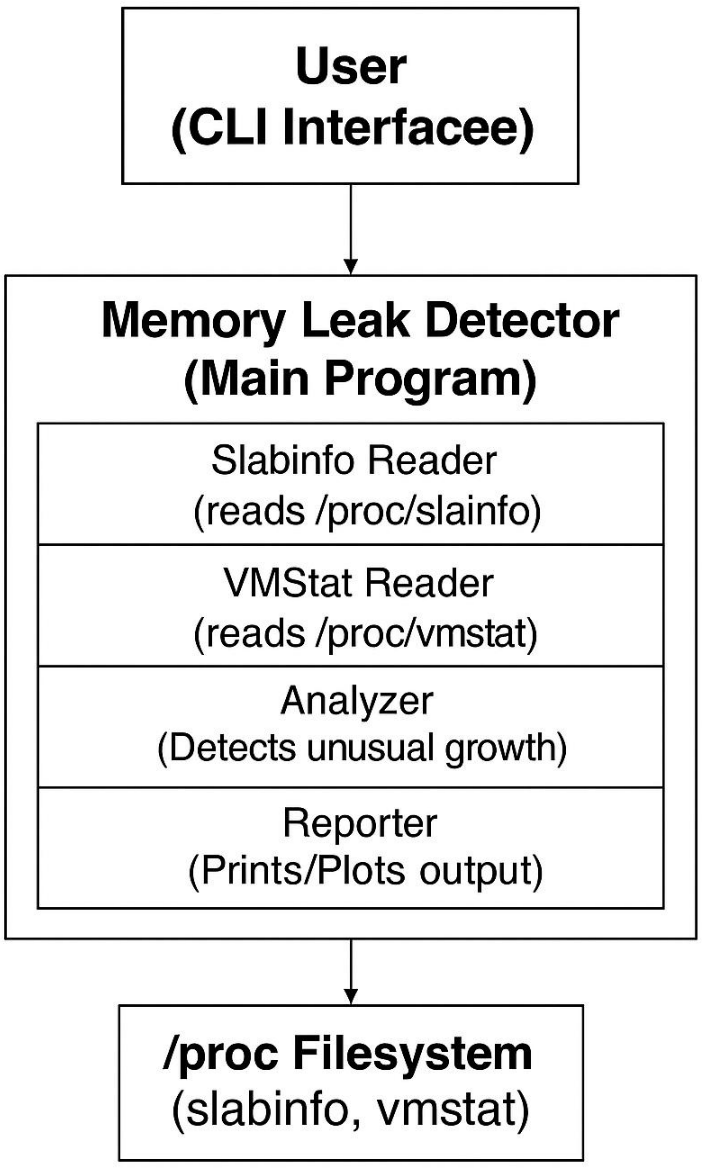


Fig.3.1. Block diagram of system architecture

- Data Reader Module:
    - Reads data from /proc/slabinfo
    - Extracts and logs relevant fields: cache_name, active_objs, total_objs
- Logger Module:

    - Appends the extracted data with timestamps into slab_log.csv
- Growth Analyzer Module:
    - Parses slab_log.csv
    - Compares multiple timestamps
    - Detects caches with consistently increasing active_objs
- Graph Generator:
    - Reads growth data
    - Displays growth using ASCII bars with optional threshold settings
- Command Line Interface:
    - User menu to select logging, analysis, view log, and show graph
    - Uses system() calls to invoke required modules

3.2 Workflow

The system follows a modular and interactive workflow designed for ease of use and clarity in memory leak detection:

1. Program Initialization via CLI

The user initiates the system by running a central Command Line Interface (CLI) tool named kernel_leak_detector. This program serves as the primary entry point for all operations.

2. User Action Selection

Upon execution, the CLI presents the user with a list of available operations — including logging slab memory info, analyzing growth, and visualizing memory trends. The user provides input to select the desired functionality.

3. Logging Slab Memory Information

If the user selects "Log Slab Info", the system runs the read_slab utility. This utility reads real-time memory slab data from the kernel's /proc/slabinfo virtual file and appends it to a CSV log file. This enables historical tracking of memory usage patterns.

4. Analysing Memory Growth

When the "Analyze Growth" option is selected, the analyze_growth component compares multiple logged snapshots of slab memory data. It identifies any significant or unusual growth in slab caches over time, potentially indicating a memory leak.

5. Graphical Visualization of Results

Choosing the “Show Graph” option invokes the show_graph module. It parses the analyzer’s output and displays a text-based bar chart in the terminal, highlighting the memory caches that are growing beyond a defined threshold.

6. Feedback Loop

The user can repeat this cycle periodically to monitor slab cache behavior, detect anomalies, and confirm whether memory issues persist or stabilize after system changes.

Chapter 4

Implementation Details

4.1 Programming Language & Platform

- Language: **C**
- OS: **Linux (Ubuntu running on VirtualBox)**
- Privilege: Requires sudo access for reading /proc/slabinfo

4.2 Key Files

- read_slab.c: Reads and logs slab data
- read_vmstat.c: Reads and displays key virtual memory statistics from /proc/vmstat
- analyze_growth.c: Analyzes growth between multiple entries
- main_cli.c: Main menu interface for user interaction
- test_all.c: Contains basic test cases for logging and analysis modules

4.3 Code highlight

This section describes the key functionalities implemented in the project:

a) Slab Info Reader

The tool accesses /proc/slabinfo to extract kernel slab cache statistics. It skips header lines and reads cache names, active objects, and total objects, which are later used for logging and growth analysis.

b) Virtual Memory Stats Reader

It reads /proc/vmstat to gather additional virtual memory metrics such as page faults, major faults, freed pages, and page steals. This offers deeper insight into overall memory behavior.

c) Timestamped Logging

Each slab cache snapshot is logged to a CSV file (slab_log.csv) along with a UNIX timestamp. This enables temporal comparison of memory usage patterns over multiple intervals.

d) Growth Analysis Logic

The analyzer parses the log file and compares slab entries between two time points. It detects caches with increasing active object counts, identifies abnormal growth, and flags them based on a set threshold.

e) Command-Line Interface (CLI)

A simple interactive CLI allows users to trigger logging, view logs, run growth analysis, display growth graphs, and exit. It improves usability and provides a menu-driven experience.

Chapter 5

Testing and Validation

5.1 Purpose

The purpose of this project is to detect potential kernel memory leaks by monitoring and analyzing the slab cache behavior using user-space tools.It aims to assist developers or system administrators in identifying unusual memory growth patterns over time.Additionally, testing and validation ensure the reliability and correctness of each module, confirming that the tool behaves as expected under various conditions.This contributes to system stability, transparency, and trust in the leak detection process.

5.2 Test cases

a. **Test Cases in read_slab (Reading & Logging Slab Info)**

Test Case ID	**Type**	**Description**	**Where It's Handled**	**Test Result**
TC-R1	Positive	File /proc/slabinfo opened successfully	fopen() check	YES
TC-R2	Negative	File not found or permission denied	if (!fp) condition	YES
TC-R3	Positive	Parsing valid lines from slabinfo	sscanf() or fscanf()	YES
TC-R4	Negative	Skip malformed or empty lines	if (field count < 3)	YES
TC-R5	Positive	Appending parsed values to slab_log.csv	fprintf() after parsing	YES
TC-R6	Positive	Read virtual memory stats (pgfault, pgmajfault, etc.)	if(pgfault, pgmajfault)	YES

b. **Test Cases in analyze_growth (Analyzing for Growth Detection**)

Test Case ID	**Type**	**Description**	**Where It's Handled**	**Test Result**
TC-A1	Positive	Open slab_log.csv successfully and read contents	fopen() check	YES
TC-A2	Negative	File not present — show error	if (!fp) condition	YES
TC-A3	Positive	Compare slab data at two time intervals	Logic in parsing loop	YES
TC-A4	Positive	Identify slab cache with actual growth	if (end > start)	YES
TC-A5	Negative	No significant growth — show appropriate message	if (growth == 0) or nothing printed	YES

c. Test cases in CLI

Test Case ID	**Covered In Code**	**Description**	**Test Result**
TC-01	CLI Option 1	Logs slab info into CSV (log created/updated)	YES
TC-02	CLI Option 3	Growth analysis runs, creates growth_log.csv	YES
TC-03	CLI Option 4	Bar graph displays based on growth_log.csv	YES
TC-04	CLI Option 2	Reads and displays contents of log file	YES
TC-05	CLI Option 5	Graceful CLI exit	YES
TC-06	File check in code	If slab_log.csv missing, error shown	YES
TC-07	File check in show_graph	Error if growth_log.csv missing	YES
TC-08	else condition in CLI	"Invalid choice" message shown	YES
TC-09	sudo required in Option 1	Without sudo, /proc/slabinfo read fails	YES
TC-10	Analyzer logic	Malformed lines are skipped or warned	YES

5.3 Testing

We performed Functional Testing (Unit-style) on each component of our kernel memory leak detection system. The goal was to ensure that each part worked correctly and integrated smoothly.

We created a test suite in C that automated the testing of all critical functionalities.

- Test 1 verified that read_slab could extract slab information and write it to a CSV file.
- Test 2 confirmed that analyze_growth correctly processed this CSV to produce meaningful output.
- Test 3 simulated invalid user input to check if the CLI handled edge cases gracefully without crashing.
- Test 4 tested the graphical display function for successful execution. Results were printed to the console with a summary of passed/failed tests.
- All components were tested in isolation and collectively, and all passed as expected, confirming the stability and reliability of our tool under standard conditions.

5.4 Methodology

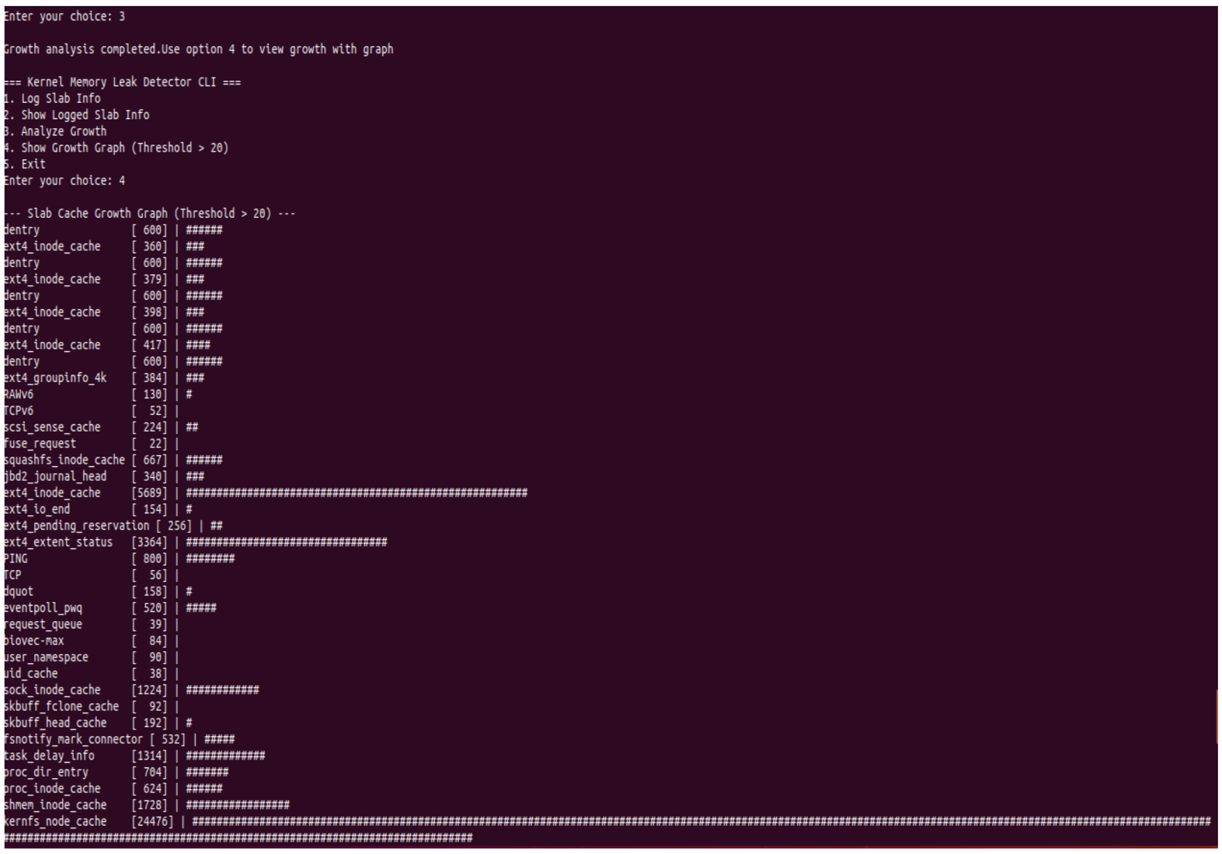
The methodology adopted for this project followed a modular, test-driven development approach. The project was implemented in C, interacting directly with Linux's virtual file system under /proc. We divided our tool into four main components: read_slab for data collection, analyze_growth for data analysis, kernel_leak_detector for CLI interaction, and show_graph for visualization. Each module was independently developed and tested for correctness and efficiency. We used functional testing to validate each binary, ensuring that the expected outputs were generated under normal and erroneous conditions. By simulating different usage scenarios, we validated robustness and functionality, all while maintaining low system overhead. We ensured the tool adheres to expected behavior in terms of input handling, system resource usage, and output formatting

Chapter 6

RESULTS

- Data was successfully logged from /proc/slabinfo using sudo ./read_slab
- Slab caches such as kmalloc-64, dentry, inode_cache showed growth during memory stress
- Graphical output accurately represented top growing slab caches
- All CLI options executed smoothly without crashing
- Thresholds effectively filtered slab growth below limits

Example Graph Output:



```
Enter your choice: 3

Growth analysis completed.Use option 4 to view growth with graph

=== Kernel Memory Leak Detector CLI ===
1. Log Slab Info
2. Show Logged Slab Info
3. Analyze Growth
4. Show Growth Graph (Threshold > 20)
5. Exit
Enter your choice: 4

--- Slab Cache Growth Graph (Threshold > 20) ---
dentry              [ 600] | ######
ext4_inode_cache    [ 360] | ###
dentry              [ 600] | ######
ext4_inode_cache    [ 379] | ###
dentry              [ 600] | ######
ext4_inode_cache    [ 398] | ###
dentry              [ 600] | ######
ext4_inode_cache    [ 417] | ####
dentry              [ 600] | ######
ext4_groupinfo_4k   [ 384] | ###
RAWv6               [ 130] | #
TCPv6               [  52] |
scsi_sense_cache    [ 224] | ##
fuse_request        [  22] |
squashfs_inode_cache [ 667] | ######
jbd2_journal_head   [ 340] | ###
ext4_inode_cache    [5689] | ########################################################
ext4_io_end         [ 154] | #
ext4_pending_reservation [ 256] | ##
ext4_extent_status  [3364] | #################################
PING                [ 800] | ########
TCP                 [  56] |
dquot               [ 158] | #
eventpoll_pwq       [ 520] | #####
request_queue       [  39] |
biovec-max          [  84] |
user_namespace      [  90] |
uid_cache           [  38] |
sock_inode_cache    [1224] | ############
skbuff_fclone_cache [  92] |
skbuff_head_cache   [ 192] | #
fsnotify_mark_connector [ 532] | #####
task_delay_info     [1314] | #############
proc_dir_entry      [ 704] | #######
proc_inode_cache    [ 624] | ######
shmem_inode_cache   [1728] | #################
kernfs_node_cache   [24476] | #######################################################################################################################################################################
#############################################################################
```

Chapter 7

CONCLUSION

This project successfully implemented a kernel memory monitoring system using only C from user-space. It bridges the gap between deep kernel diagnostics and user-space accessibility by:

- Reading slab memory stats from virtual files
- Logging and analyzing for trends
- Generating simple and effective text-based visualizations

The project is extensible and forms a good base for future work involving real-time alerts, integration with monitoring dashboards, or even graphical UI.

Key areas focused:

Concept	Status	Explanation
1. Memory Management	Covered	Actively monitored kernel memory (slab caches) via /proc/slabinfo.
2. Virtual Memory	Covered	/proc/vmstat uses virtual filesystem reflecting kernel data, including memory stats.
3. Processes	Covered (via CLI)	You ran multiple C programs as separate processes (read_slab, analyze_growth).
4. File System	Covered	Logged data to slab_log.csv, read and analyzed files using C File I/O.

Chapter 8

LEARNING OUTCOMES

- Gained hands-on understanding of Linux internals, especially kernel memory management via the /proc interface.
- Learned to parse kernel virtual files and analyze slab memory allocation behavior in Linux.
- Practiced file I/O operations and string parsing using C.
- Designed and implemented modular C programs for different functionalities.
- Built simple graph generation utilities to visualize kernel memory growth in the terminal.
- Developed and tested Command Line Interface (CLI) tools to interact with system components.
- Wrote and executed functional (unit-style) test cases to verify system behavior.
- Understood how to debug privilege-related and access issues in a Linux environment.
- Strengthened teamwork through collaboration, version control, and communication during development.
- Improved our ability to design, develop, test, and document a multi-component system - a key skill for real-world engineering projects.

Chapter 9

REFERENCES

[1] J. Bonwick, *The Slab Allocator: An Object-Caching Kernel Memory Allocator*, Sun Microsystems, 1994. [Online]. Available: Uploaded as bonwick_slab.pdf.

[2] P. Jain and S. Sehgal, */proc File System in Linux*, Presentation Slides. [Online]. Available: Uploaded as PROC.ppt.

[3] "Linux vmstat command," [Online]. Available: Uploaded as 28175285-Linux-vmstat-command.pdf.

[4] D. Marinas, *Tracking Kernel Memory Leaks using KMEMLEAK*, LinuxCon Europe, 2011. [Online]. Available: Uploaded as lceu11_marinas.pdf.

[5] A. Lakshmanan et al., *k-meld: A Kernel Memory Leak Detector*, 2009. [Online]. Available: Uploaded as k-meld.pdf.

[6] LabEx, "How to Check if a Specific Kernel Slab Setting is Active in Linux," *LabEx.io Tutorials*, [Online]. Available: https://labex.io/tutorials/linux-how-to-check-if-a-specific-kernel-slab-setting-is-active-in-linux-558759 [Accessed: Jul. 28, 2025].

[7] Clever Uptime Docs, "/proc/slabinfo: Explanation & Insights," *Clever Uptime*, [Online]. Available: https://cleveruptime.com/docs/files/proc-slabinfo [Accessed: Jul. 28, 2025].

[8] B. Gregg, "Linux Performance Analysis - Understanding vmstat," *YouTube*, Oct. 2016. [Online Video]. Available: https://www.youtube.com/watch?v=j-yDDfmtGQc [Accessed: Jul. 28, 2025].

[9] NetworkChuck, "Linux Sysadmin Basics – The /proc Filesystem," *YouTube*, Nov. 2020. [Online Video]. Available: https://www.youtube.com/watch?v=0XdjODvsRN8 [Accessed: Jul. 28, 2025].

[10] Kaiwan N Billimoria, "procmap: Visualizing Kernel and Userspace Memory Maps," *GitHub Repository*, 2023. [Online]. Available: https://github.com/kaiwan/procmap/blob/master/README.md [Accessed: Jul. 28, 2025].