

# Persistent Memory Leak Detector

Operating Systems (Course Code: CS303)

## Group Members:

(202351129) Ram Sharma

(202351137) Kush Sonawane

(202351164) Charvi Sri Sravanam

Indian Institute of Information Technology, Vadodara  
Gandhinagar Campus

October 2025

# Problem Statement, Motivation & Objectives

## Problem Statement:

Memory leaks in long-running programs cause gradual performance degradation. Traditional tools detect leaks only during a single run and lose data afterward.

**Goal:** Design a detector that logs and persists leak data for long-term analysis.

## Motivation:

Memory leaks are difficult to trace in complex, long-running software such as servers or background daemons. Conventional tools lose information after each execution, making pattern detection and debugging harder.

## Real-World Applications:

- Used in large-scale backend systems (e.g., web servers, database engines) for monitoring memory health.

## Objectives:

- Detect dynamically unfreed memory during runtime.
- Store leak information in a persistent SQLite database.
- Compare results across multiple executions.
- Improve reliability and reduce debugging effort.

# Methodology

## Phase 1 – Monitoring

- Hook malloc/free using LD\_PRELOAD.
- Record address, size, PID, timestamp.

## Phase 2 – Persistence & Analysis

- Log data into memory\_leak.db.
- Compare leak reports across runs via Python script.

# System Setup

- OS: Ubuntu 24.04 LTS, GCC 13, SQLite 3.45
- Language: C + Python
- Run: LD\_PRELOAD=./memory\_hook.so ./test\_app
- Analyze: python3 analyze\_leaks.py summary

# System Architecture

## Overview:

The system architecture shows how all components of the **Persistent Memory Leak Detector** interact with each other and the Operating System to detect, log, and analyze memory leaks persistently.

## Component Roles:

- **memory\_hook.c** — Intercepts `malloc()` and `free()` calls using `LD_PRELOAD` and logs them to the database.
- **memory\_server.c** — Acts as the user program whose memory allocations and deallocations are being monitored.
- **analyze\_leaks.py** — Compares and reports memory leaks across multiple program runs.
- **live\_graph.py** — Displays real-time visualization of memory usage and leak growth.
- **memory\_leak.db** — SQLite database storing all logged allocation.

## Control Flow:

```
memory_server.c → memory_hook.c → memory_leak.db →  
analyze_leaks.py / live_graph.py
```

# Implementation

## C Layer

- Hooks into `malloc/free` and logs to SQLite.
- Thread-safe with `pthread_mutex`.

## Python Layer

- Generates summaries, comparisons, and JSON reports.

# Runtime Output

Execution showing memory hook initialization and detected allocations.

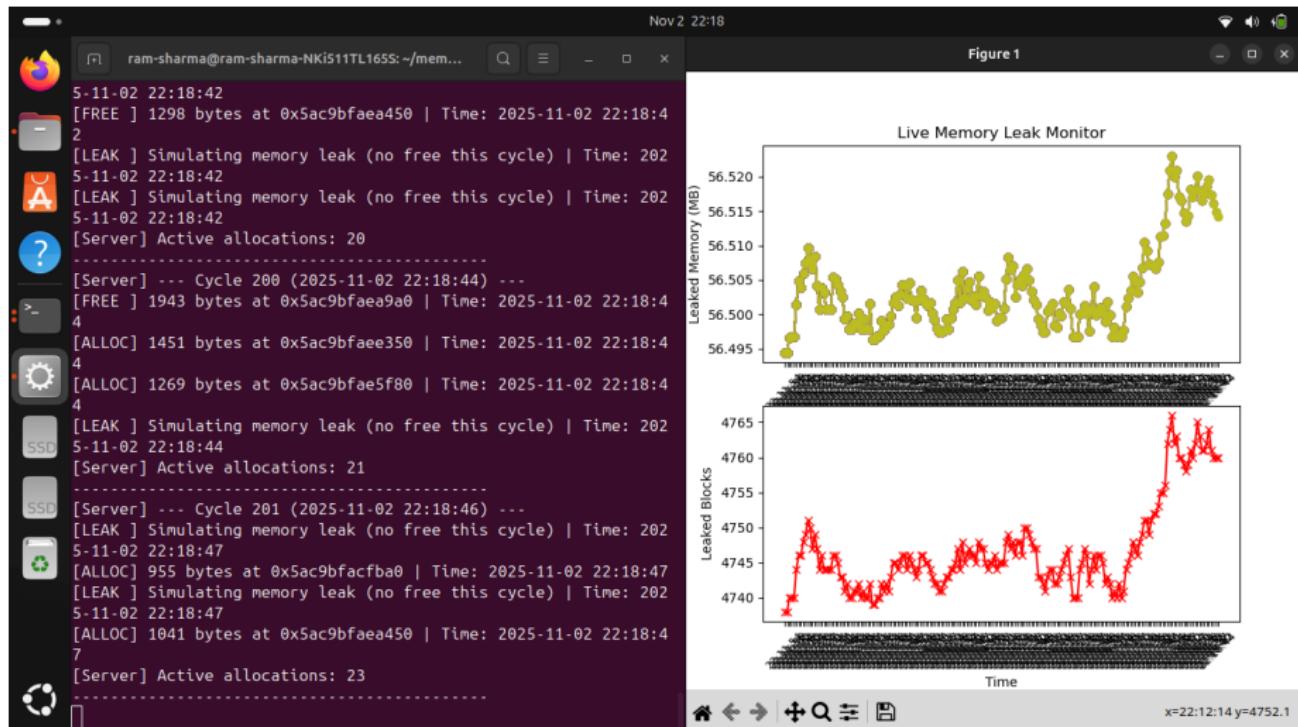


Figure: Terminal showing memory hook initialization + allocations

# Leak Summary Analysis

SQLite output summarizing total leaked allocations and memory bytes.

```
Nov 2 23:21
ram-sharma@ram-sharma-NKi511TL165S: ~/memory_leak_detector
[Server] Process stopped. Data persisted to DB (memory_leak.db).
[Server] Use 'python3 analyze_leaks.py summary' to view report.
[memory_hook] Cleaning up...
ram-sharma@ram-sharma-NKi511TL165S: ~/memory_leak_detector$ python3 analyze_leaks.py
=====
MEMORY LEAK DETECTOR - ANALYSIS SUMMARY
=====
Current Status:
Total Leaked Allocations: 4762
Total Leaked Memory: 59262598 bytes
Average Leak Size: 12444.90 bytes

Leak History (Last 10):
No historical data yet

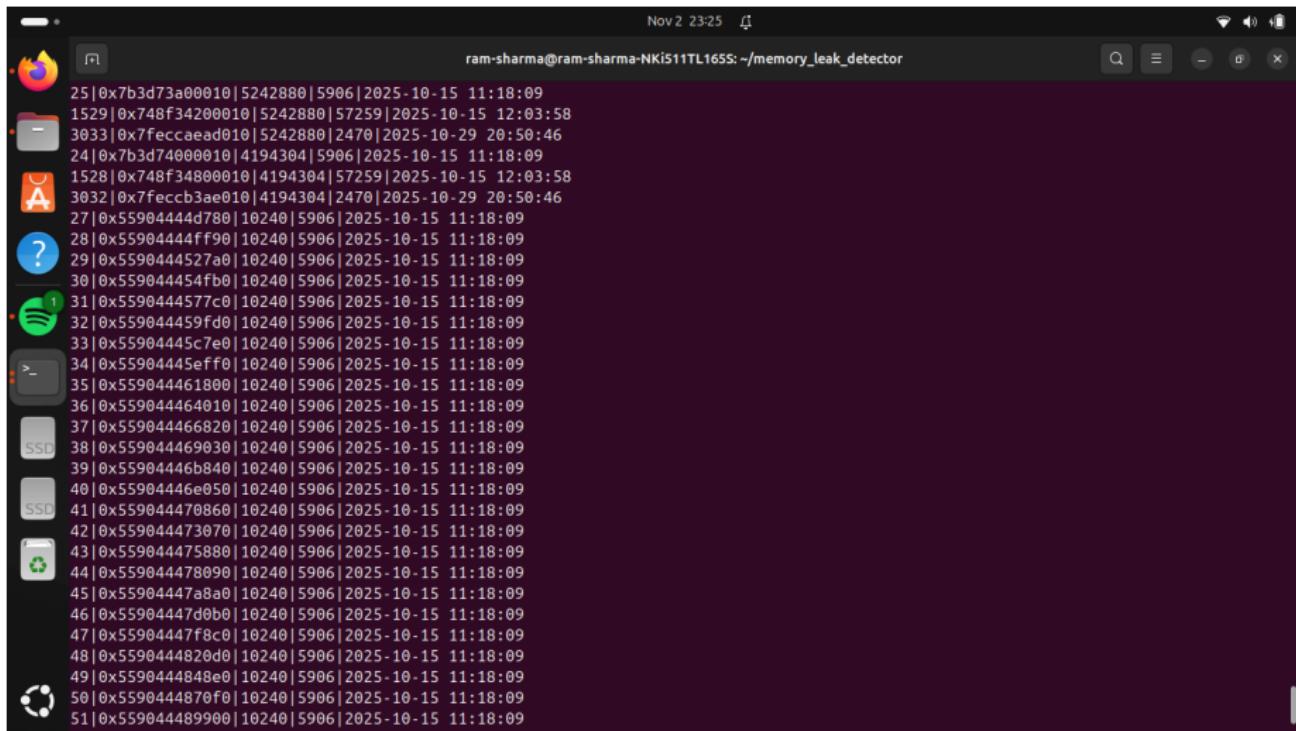
SSD Top Leaks (by size):
1. Address: 0x7b3d73a00010, Size: 5242880 bytes
2. Address: 0x748f34200010, Size: 5242880 bytes
3. Address: 0x7feccaead010, Size: 5242880 bytes
4. Address: 0x7b3d74000010, Size: 4194304 bytes
5. Address: 0x748f34800010, Size: 4194304 bytes

=====
ram-sharma@ram-sharma-NKi511TL165S: ~/memory_leak_detector$ sqlite3 memory_leak.db
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
sqlite> .allocations
Error: unknown command or invalid arguments: "allocations". Enter ".help" for help
sqlite> .tables
```

Figure: SQLite query result and summary table from analysis

# Persistent Leak Database

Historical leak data stored in `memory_leak.db` across multiple runs.



The screenshot shows a terminal window on a Linux desktop environment. The title bar reads "ram-sharma@ram-sharma-NKi511TL165S: ~/memory\_leak\_detector". The terminal window displays a list of 51 rows of data, each representing a memory leak entry. The columns are separated by vertical pipes (|). The first column contains row numbers (e.g., 25, 51). The second column contains memory addresses (e.g., 0x7b3d73a00010, 0x5590444848e0). The third column contains pointers (e.g., 5242880, 10240). The fourth column contains file offsets (e.g., 5906, 5906). The fifth column contains timestamps (e.g., 2025-10-15 11:18:09, 2025-10-15 11:18:09).

25	0x7b3d73a00010	5242880	5906	2025-10-15 11:18:09
26	0x748f34200010	5242880	57259	2025-10-15 12:03:58
27	0x7feccaead010	5242880	2470	2025-10-29 20:50:46
28	0x7b3d74000010	4194304	5906	2025-10-15 11:18:09
29	0x748f34800010	4194304	57259	2025-10-15 12:03:58
30	0x7feccb3ae010	4194304	2470	2025-10-29 20:50:46
31	0x559044444d780	10240	5906	2025-10-15 11:18:09
32	0x55904444ff9f0	10240	5906	2025-10-15 11:18:09
33	0x5590444527a0	10240	5906	2025-10-15 11:18:09
34	0x559044454fb0	10240	5906	2025-10-15 11:18:09
35	0x5590444577c0	10240	5906	2025-10-15 11:18:09
36	0x559044459fd0	10240	5906	2025-10-15 11:18:09
37	0x55904445c7e0	10240	5906	2025-10-15 11:18:09
38	0x55904445eff0	10240	5906	2025-10-15 11:18:09
39	0x559044461800	10240	5906	2025-10-15 11:18:09
40	0x559044464010	10240	5906	2025-10-15 11:18:09
41	0x559044466820	10240	5906	2025-10-15 11:18:09
42	0x559044469830	10240	5906	2025-10-15 11:18:09
43	0x55904446b840	10240	5906	2025-10-15 11:18:09
44	0x55904446e050	10240	5906	2025-10-15 11:18:09
45	0x559044470860	10240	5906	2025-10-15 11:18:09
46	0x559044473070	10240	5906	2025-10-15 11:18:09
47	0x559044475880	10240	5906	2025-10-15 11:18:09
48	0x559044478090	10240	5906	2025-10-15 11:18:09
49	0x55904447a8a0	10240	5906	2025-10-15 11:18:09
50	0x55904447db00	10240	5906	2025-10-15 11:18:09
51	0x55904447f8c0	10240	5906	2025-10-15 11:18:09

Figure: Database table view showing persistent stored data

# References

- Linux Man Pages — LD\_PRELOAD
- SQLite Documentation — <https://www.sqlite.org>
- Valgrind Developers, Memcheck: A Memory Error Detector, 2024.  
Available: <https://valgrind.org/docs/manual/mc-manual.html>  
→ Conceptual comparison for professional memory leak detection methodology.