# Memory Leak Detection Tool

:intercepts memory allocation (malloc, calloc, realloc) and deallocation (free) system calls, report which have not been freed

Mudadla Sai Prasanna : 2203120
Maryada Harshini : 2203118
Venkatapuram Pooja : 2203138

# Overview of memory leaks

- **Importance of Precision in Software Development**: Precision is critical in software development, especially in memory management.
- **Impact of Memory Leaks:** Memory leaks cause performance issues and waste resources
- **Need for Advanced Tools:** Advanced tools help developers detect ad fix memory leaks effectively.
- **Objective:** Ensure robust applications with optimal performance

# Importance of memory leak detection

- **Significance of Memory Leak Detection:** Essential for maintaining application performance and reliability.
- **Early Detection Benefits:** Prevents crashes, slowdown, and saves time and resources.
- **Impact on Software Quality:** Enhances software quality and ensures seamless, efficient operation.
- **User Satisfaction:** promotes a better user experience with smooth application performance.
- **Future Readiness:** Supports innovation and lays a strong foundation for future projects.

# What is Memcheck?

- A memory leak detection tool that intercepts memory allocation (malloc, calloc, realloc) and deallocation (free) system calls.
- Reports memory that has not been freed, helping detect memory leaks.
- **Intercepts system calls**:Memcheck intercepts calls to **malloc, calloc, realloc, valloc, memalign, free, new, and delete.**
- Reports detailed information:
  - Amount of memory leaked.
  - Stack trace leading to the allocation.

# Alternatives to Memcheck

- Various tools are available to assist developers in identifying memory leaks efficiently.
- **Tools to Detect Memory Leak**:Some are Dmalloc, Electric Fence, Dbgmem, Memwatch, Mpatrol .
- Popular options include Valgrind, which provides detailed memory usage analysis, and AddressSanitizer, known for its speed and integration with modern compilers.
- **Additional Tools:** Visual Studio's built-in diagnostics and Eclipse Memory analyzer enhance the detection process, enabling developers to pinpoint leaks with precision
- Leveraging these tools is crucial for maintaining high-quality applications

# Custom Memory Management Tool

- Tracks malloc, calloc, realloc, and free calls.
- Reports memory that hasn’t been freed.
- Aids in effectively detecting memory leaks.

# Why Use a Memory Leak Detection Tool?

- Prevents memory leaks and improves software reliability.
- Tracks and reports detailed memory usage.
- Simplifies debugging of memory management issues.

# Project File Components

- **leak.h**:
  - Defines structures and functions for memory tracking.
  - Redefines standard malloc, calloc,realloc and free functions to custom versions that log memory operations.
- **example.c**:
  - A sample program to demonstrate memory allocation and deallocation.
  - Contains intentional memory leaks for testing the tool's reporting capabilities.
- **test.c**:
  - Uses the utest framework to run unit tests(if required).
- **Makefile**:
  - Compiles the project and creates the example binary for testing
  - Provides a clean target to remove compiled files.
- **leak_detector.c:**
  - Tracks memory allocated and freed during the execution of a program.
  - Reports any memory that was not freed(leaks)

# Functional overview of leak.h

- **Custom Allocation Functions:**
  - _malloc, _calloc, _realloc: Replace standard memory allocation functions to track and log memory usage.
  - _free: Tracks and validates memory deallocation.
- **Tracking Operations:**
  - **_insert:** Records each memory allocation with size, address, file, and line number.
  - **_delete:** Removes allocation records when memory is freed.
- **Report Generation:**
  - **_generate_report:** Summarizes memory usage, allocations, and leaks at program termination.
- **Warnings:**
  - **_leak_warn:** Prints warnings for invalid operations like double frees or failed allocations.

- **Data Structures Used:**
  - **Array-Based Tracking:** Maintains records of memory blocks up to a predefined limit (LEAK_MEM_SIZE).
- **Detection Logic:**
  - Compare allocation addresses to identify leaks or redundant frees.
  - Calculate leaked memory and generate percentage-based summaries.

# How to Use the Tool

- Ensure the files **leak.h, example.c, test.c,leak_detector.c** and **Makefile** are in the same project root directory.
- Run the following commands:
  - Compile the project using the command: **make all**
    - This will compile **example.c** and create an executable named **example**.
  - Run the program using the command: **./example**
    - The program will excute, and atexit() will call generate_report() at the end of the report memory that hasn't been freed.
  - Compile and run test.c uisng the Makefile target using the command: **make runtest**.

- After running the Program we get the following output:

```
========= MEMORY LEAK DETECTOR REPORT =========
Total Allocations:      5
Total Frees:            3
Total Memory Allocated: 325 bytes
Total Memory Freed:     125 bytes
Memory Leaked:          200 bytes
Leak Percentage:        61.54%

======== LEAK DETAILS ========
Leaked Memory:
  - Address:  0x555f5202a310
  - Size:     120 bytes
  - Location: example.c:8
Leaked Memory:
  - Address:  0x555f5202a390
  - Size:     80 bytes
  - Location: example.c:11
==============================================
```

```c
char *b = malloc(120);
free(b);                    //Here memory will be freed
```

- After adding a new line "free(b)" to example.c file ,we get the following output:

```
========= MEMORY LEAK DETECTOR REPORT =========
Total Allocations:      5
Total Frees:            4
Total Memory Allocated: 325 bytes
Total Memory Freed:     245 bytes
Memory Leaked:          80 bytes
Leak Percentage:        24.62%

======== LEAK DETAILS ========
Leaked Memory:
  - Address:  0x564d06681390
  - Size:     80 bytes
  - Location: example.c:12
==============================================
```

# Best practices for prevention

- **Regular Code Reviews**:

Conduct peer reviews to identify and rectify potential memory management issues.

- **Utilize Automated Testing Tools**:

Leverage tools like Valgrind, AddressSanitizer, or other memory analyzers to detect leaks early.

- **Encourage Modular Programming**:

Design code in smaller, independent modules to isolate and identify sources of memory leaks efficiently.

- **Implement Memory Profiling**:

Use memory profiling tools during development to monitor and analyze memory usage patterns.

- **Integrate Leak Prevention in Workflow**:

Make memory management a part of the development process to enhance application stability and performance.

# References

https://www.geeksforgeeks.org/how-to-detect-memory-leaks-in-c/

https://www.tutorialspoint.com/what-is-dynamic-memory-allocation-in-c

https://github.com/sheredom/utest.h/blob/master/utest.h

# THANK YOU