RV College of Engineering®

Go, change the world

Experiential Learning
Phase 1 & 2: Presentation

Topic: Memory management API

SEMESTER	3
BRANCH	COMPUTER SCIENCE ENGINEERING

TEAM INTRODUCTION

Roll No.	USN	Name	Email Id
1.	1RV22CS154	Raghuveer Narayanan Rajesh	raghuveernr.cs22@rvce.edu.in
2.	1RV22CS175	Samvit Sanat Gersappa	samvitsanatg.cs22@rvce.edu.in
3.	1RV22CS153	Priyansh Rajiv	tarunbhupathi.cs22@rvce.edu.in
4.	1RV22CS162	Rohan J S	rohanjs.cs22@rvce.edu.in

Literature Review

AUTHOR	PAPER TITLE	PUBLICATION DETAILS	SUMMARY	REMARKS
• Karthikeyan, • V., S. Ravi • M. Anand	Robust memory management using real time concepts	Journal of computer science	• The paper starts by introducing the various aspects of memory management and virtual memory like fragmentation, paging etc. • It explains the problem of fragmentation of memory when dynamic memory allocation is used on certain operating systems • It provides a solution to solve the fragmentation process by allocating pools of memory	• The problem of fragmentation can be solved by allocating fixed sized blocks of memory during compile time to a process. • This strategy, if used efficiently, can improve memory consumption and time overhead.
• Ben Kenwright	Fast Efficient Fixed-Size Memory Pool: No Loops and No Overhead	School of Computer Science Newcastle University Newcastle, United Kingdom,	• The paper starts with explaining the time overhead of inefficient memory management techniques. • It proposes an efficient memory management technique of memory pools. • It evaluates the advantages and limitations of the pool allocating	• It proposes a memory pool allocation system with a link to the first free and first occupied node for efficiency. • It proposes dynamic changing of the block size. • The limitation of this system is if the allocated memory are of very small sizes compared to the block size and unpredictability when used in multithreaded applications.

Literature Review

AUTHOR	PAPER TITLE	PUBLICATION DETAILS	SUMMARY	REMARKS
• Manish Pandey • Young Woo Kwon	Optimizing Memory Allocation in a Serverless Architecture through Function Scheduling	2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)	• This paper proposes a machine learning approach to optimize memory allocation for microservices in a serverless architecture. • It addresses the issue of over-allocation and its negative impact on resource utilization and waste. • The authors propose a fine-grained scheduling method that estimates memory usage and schedules similar functions on the same node, reducing overall node utilization and keeping data local.	• Analyze past API calls to predict memory demands for different functionalities. • Based on these predictions, group or prioritize requests to achieve efficient resource utilization, potentially avoiding over-allocation and improving overall API performance. • This approach requires adapting the specific algorithms and data structures to your unique API design and usage patterns, but the underlying idea of data-driven memory management remains applicable.

Literature Review

AUTHOR	PAPER TITLE	PUBLICATION DETAILS	SUMMARY	REMARKS
• Jiaheng Wu, • Michael Whittaker, • Yixin Sun	Scalable, Efficient, and Flexible User-Level Memory Management for C	Proceedings of the 31st ACM SIGPLAN International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '22)	• This paper proposes a new user-level memory management API for C, offering scalability, efficiency, and flexibility compared to traditional malloc/free.	• This paper looks at alternative memory management solutions for performance critical C applcations
• Xinchen Guo, • Shubin Xie, • Yuanyuan Feng, • Haibo Chen	Unified Memory Management Model for C Operating Systems	arXiv 2023	• This paper proposes a unified memory management model for C operating systems that combines static and dynamic allocation while ensuring safety and efficiency. • The model aims to offer flexibility and control while avoiding the pitfalls of each individual approach.	• This paper presents a new conceptual model, but further research and evaluation are needed to assess its viability and practical implementation in real-world operating systems.

Literature Review

AUTHOR	PAPER TITLE	PUBLICATION DETAILS	SUMMARY	REMARKS
• Yang Xiao, • Chengjie Wu, • Wenchao Li, • Gang Zhou, • Jisheng Zhao	• Optimizing Memory Allocation in Real-time Systems with C++-like Smart Pointers in C (RTAS 2023)	RTAS 2023	• Investigates benefits of using C++-like smart pointers in C for real-time systems, proposing modified versions optimized for C to reduce overhead and offer safe, automatic memory management.	• Practical approach for improved memory safety and potential performance gains in real-time systems, but requires code modification and might not suit all constraints.

Relevance to the course

Memory Management:

Memory allocation and deallocation are fundamental aspects of operating systems. Your API's seamless allocation and deallocation of memory are directly related to memory management techniques taught in operating systems courses, such as dynamic memory allocation algorithms, memory fragmentation, and memory leak detection.

API Design and Implementation:

Developing an API for memory management involves designing interfaces and functions that enable users to allocate and deallocate memory seamlessly. We utilize API design principles, including abstraction, encapsulation, and modularity. We discuss APIs are implemented and how they interact with underlying system resources.

System Calls and Interface:

APIs often serve as an interface between user-level applications and the operating system kernel. In an operating systems course, students learn about system calls, which are mechanisms for invoking operating system services from user-space applications. Understanding how APIs interact with system calls and the kernel is crucial for designing efficient and secure software systems.

PROBLEM STATEMENT

The problem entails designing and implementing a memory management API to facilitate users in efficiently utilizing system memory. This API should enable seamless allocation and deallocation of memory while optimizing resource utilization. Additionally, thorough testing using appropriate data structures is essential to validate the functionality and robustness of the API.

OBJECTIVES

- To enable users to easily access and store relevant memory management parameters and compare it with future runs to test memory efficiency.
- To develop functions to handle create, deletion and manipulation of memory pools to enable access of memory minimizing overhead.
- To test the developed functions with data structures and algorithms.

METHODOLOGY OUTLINE

APIs used

1. sys.resource.h: getrusage() function from this header file returns number of pages replaced and context switches in long int form and stores it in structure of type rusage.

2. Proc filesystems: Proc filesystems can be parsed to retrieve the following memory usage parameters of the running process:
   - Program size
   - Resident set size
   - Shared memory
   - Text memory
   - Library memory
   - Dirty pages

3. stdlib.h: For standard library memory allocation and deallocation systems like malloc() and free().

4. stddef.h: For using data types like size_t that are required to handle memory management.

5. stdio.h: File functions like fscanf(), fprintf(), fread(), fwrite(), fclose() to store memory data.

Applications

1. Memory-Intensive Applications: Programs dealing with large datasets, complex simulations, or high-performance computing could utilize the API for custom memory management strategies, tailoring allocations for specific use cases.
2. Secure Coding: An API could offer features for memory-safe programming, preventing common vulnerabilities like buffer overflows and memory leaks, improving application security and stability.
3. Real-Time Systems: Real-time systems demanding predictable performance could use specific APIs designed for time-sensitive memory operations, guaranteeing timely responses and avoiding memory-related delays.
4. High-Performance Computing: In HPC environments, APIs could manage distributed memory across clusters, optimizing communication and ensuring efficient parallelization for large-scale computations.
5. System Design: Memory management APIs could be included in system design tools, allowing developers to model and predict memory usage scenarios, leading to more efficient and scalable systems.

Thank You