**Software Requirements Specification**

For

Dynamic Memory Manager in Linux using C language

11th November 2021

Prepared by

|  |  |  |
| --- | --- | --- |
| **Specialization** | **SAP ID** | **Name** |
| AI&ML | 500075765 | Ansh Gupta |
| CCVT | 500076357 | Vitesh Sethi |
| AI&ML | 500075858 | Aryan Rastogi |



Department of Informatics

School Of Computer Science

UNIVERSITY OF PETROLEUM & ENERGY STUDIES,

DEHRADUN- 248007. Uttarakhand

Table of Contents

|  |  |  |
| --- | --- | --- |
| **Topic** | | **Page No** |
| Table of Content | |  |
| Revision History | |  |
| 1 | Introduction | 4 |
|  | 1.1 Purpose of the Project | 4 |
|  | 1.2 Target Beneficiary | 4 |
|  | 1.3 Project Scope | 5 |
|  | 1.4 References | 5 |
| 2 | Project Description | 5-6 |
|  | 2.1 Reference Algorithm | 6-7 |
|  | 2.2 Data/ Data structure | 7 |
|  | 2.3 SWOT Analysis | 7 |
|  | 2.4 Project Features | 8 |
|  | 2.5 Design and Implementation Constraints | 8 |
|  | 2.6 Design diagrams | 8 |
| 3 | System Requirements | 9 |
|  | 3.1 User Interface | 9 |
|  | 3.2 Software Interface | 9 |
|  | 3.3 Database Interface | 9 |
|  | 3.4 Protocols | 9 |
| 4 | Non-functional Requirements | 9 |
|  | 4.1 Performance requirements | 9 |
|  | 4.2 Security requirements | 9 |
|  | 4.3 Software Quality Attributes | 9 |
| 5 | Other Requirements | 9 |
| Appendix A: Glossary | | 10 |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Change** | **Reason for Changes** | **Mentor Signature** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. INTRODUCTION

**Memory Allocation** is a process by which a user-space process/application is allocated or deallocated with physical/virtual memory space by the **Kernel** Memory Management Unit. This allocation can be done in two ways that are **Static** and **Dynamic**. In static memory allocation, the memory requested by the user-space process is known and memory is allocated in the **Stack** virtual memory. In dynamic memory allocation, the memory requested by the user-space process is unknown and memory is allocated in the **Heap** virtual memory. In stack, memory is allocated and deallocated by the compiler itself while in heap the programmer has to do perform allocations and deallocations manually. Hence, the handling of the heap becomes costlier than that of the stack. One of the most significant issues faced in dynamic memory allocation is **Memory Fragmentation**. The creation of very small inefficient memory spaces(holes) in the virtual memory space due to continuous allocation and deallocation of memory is known a memory fragmentation. In the C programming language, the dynamic memory is handled by standard **Glibc** library functions such as **malloc()** and **free()**. In this project, a custom memory manager is built in Linux OS using system calls such as **mmap()** and **munmap()**. The proposed technique shows the detailed statistics of memory in use via every structure that helps in catching **Memory Leaks**. Memory leaks occur when the memory used by dynamically created objects is not deallocated after their use. The proposed memory manager uses **xmalloc()** function to allocate the memory and **xfree()** to deallocate the memory.

1.1 Purpose of the Project

In this project, a customized **Dynamic** Memory Manager/**Heap** Memory Manager is built in Linux operating system using C language. The main aim of the memory manager is to allocate and deallocate memory as per the user’s requirement. The proposed approach shows the statistics of memory in use by every structure. The required modifications will be performed over the memory manager so that the problem of memory fragmentation can be addressed. The proposed technique will demonstrate the internal design and implementation of the standard Glibc library functions malloc() and free().

1.2 Target Beneficiary

The target beneficiaries of the proposed technique are programming enthusiasts and technical companies working in the system programming domain. They do not use the Glibc library functions malloc() and free() directly for memory allocation in their applications. They build their own custom memory manager on top of malloc() and free() function calls. In this project, a similar kind of memory manager will be developed that can be used by the students, programmers and technical companies in their upcoming initiatives.

1.3 Project Scope

The scope our project is to replace the standard Glibc library with the proposed memory manager. The standard malloc() and free() functions will be replaced by xmalloc() and xfree() respectively for the allocation and deallocation of memory. The memory requested by the user is allocated in multiples of virtual memory pages. This project is very useful for the students and tech enthusiasts who are working in the domain of memory management. The project provides the detailed information about the logics of operating system related to virtual memory of the system.

1.4 References

[1] E. D. Berger, K. S. McKinley, R. D. Blumofe, and P. R. Wilson, “Hoard: a scalable memory allocator for multithreaded applications,” ACM SIGARCH Computer Architecture News, vol. 28, no. 5, pp. 117–128, 2000..

[2] J. Evans, “A Scalable Concurrent malloc(3) Implementation for FreeBSD,” 2006.

[3] “TCMalloc : Thread-Caching Malloc.” http://goog-perftools.sourceforge.net/doc/tcmalloc.html (accessed Oct. 25, 2021).

[4] “jemalloc.” http://jemalloc.net/ (accessed Oct. 05, 2021).

[5] “Scalable memory allocation using jemalloc - Facebook Engineering.” https://engineering.fb.com/2011/01/03/core-data/scalable-memory-allocation-using-jemalloc/ (accessed Oct. 05, 2021).

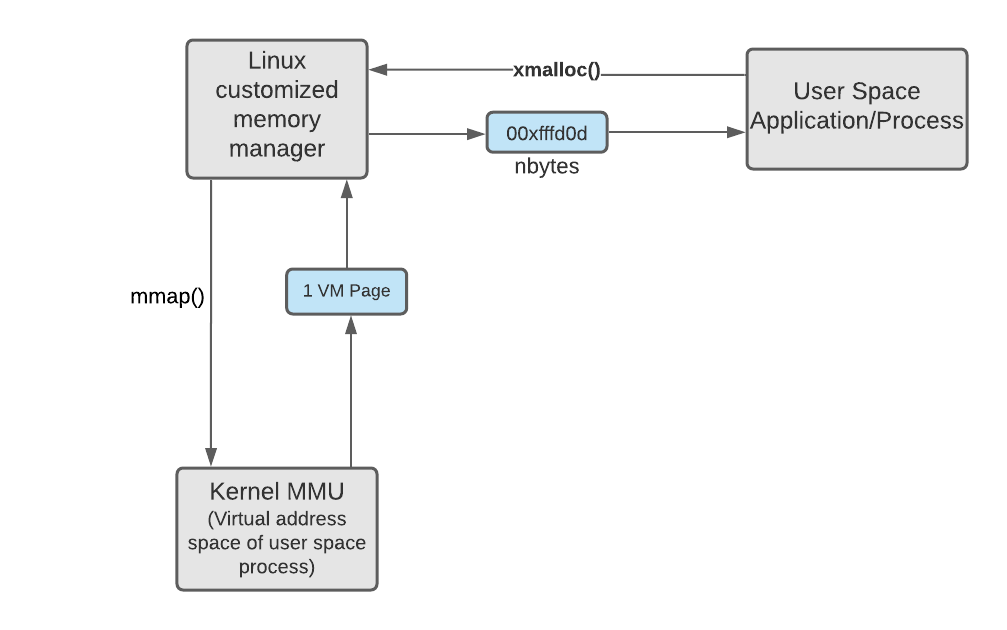
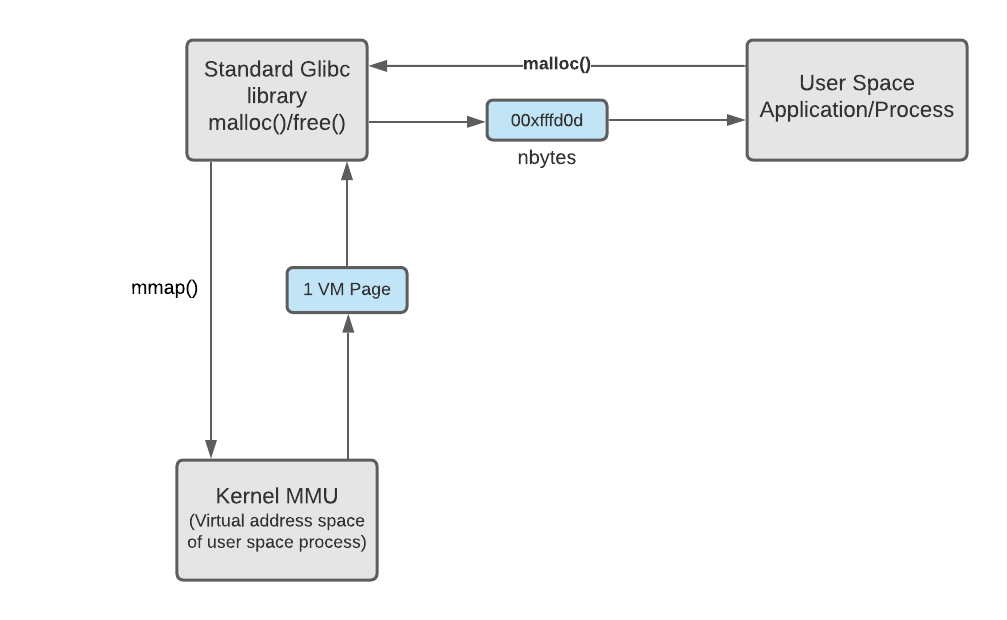
[6] E. D. Berger, B. G. Zorn, and K. S. McKinley, “Reconsidering custom memory allocation,” ACM SIGPLAN Notices, vol. 37, no. 11, pp. 1–12, 2002.

2. PROJECT DESCRIPTION

The overall design of memory management consists of three components :-

* User Space Process/Application - A standard C program can serve as a user-space process/application. This user-space process makes requests to the memory manager to allocate and deallocate n bytes of memory as per its requirement.
* Standard Glibc library- Memory allocation and deallocation between Glibc and Kernel Memory Management Unit happen only in units of page sizes via mmap() system calls. Standard Glibc library caches the Virtual Memory (VM) Page allocated by Kernel MMU and allocates chunks of it to user space process. When the user space process requests to deallocate memory, the standard Glibc library returns the VM Page back to the Kernel MMU via munmap() system calls.
* Kernel Memory Management Unit - It provides Virtual Memory Pages to the standard Glibc library as per the request made by it.

Fig 1 describes the standard Glibc memory management. The user will perform the read/write operations over the memory allocated through virtual memory pages.

 Fig 1: Standard memory manager Fig 2: Customized memory manager

In the proposed mechanism shown in Fig 2, the standard Glibc library is replaced with customized Linux Memory Manager. In other words, another version of malloc() and free() operations are built that are xmalloc() and xfree() respectively.

2.1 Reference Algorithm

The main algorithm of our project comprises of the given below steps:

Fig 3 describes the step by step process of allocation of memory through proposed approach. Memory is allocated only if kernel finds free available memory.

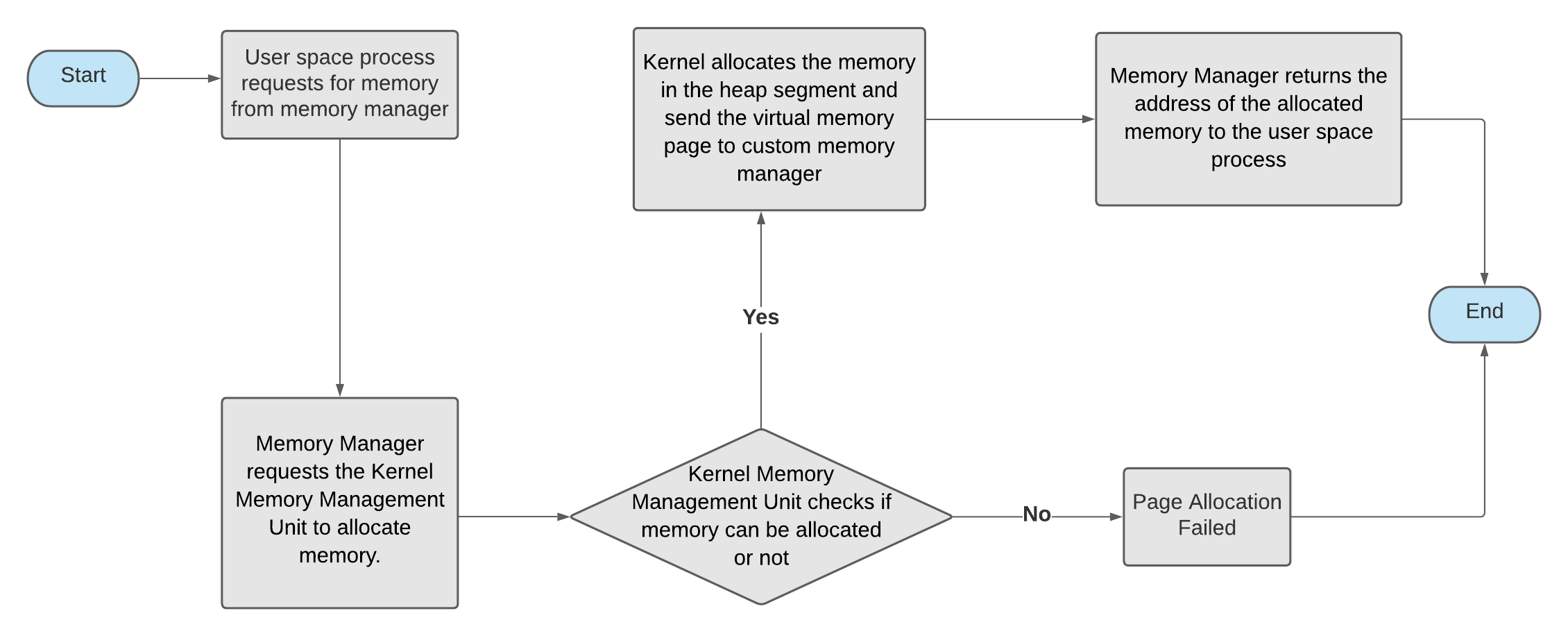


Fig 3: Algorithm for allocation of memory

Fig 4 describes the step by step process of de-allocation of memory through proposed approach. Memory is de-allocated only if kernel finds that memory address is valid or not.

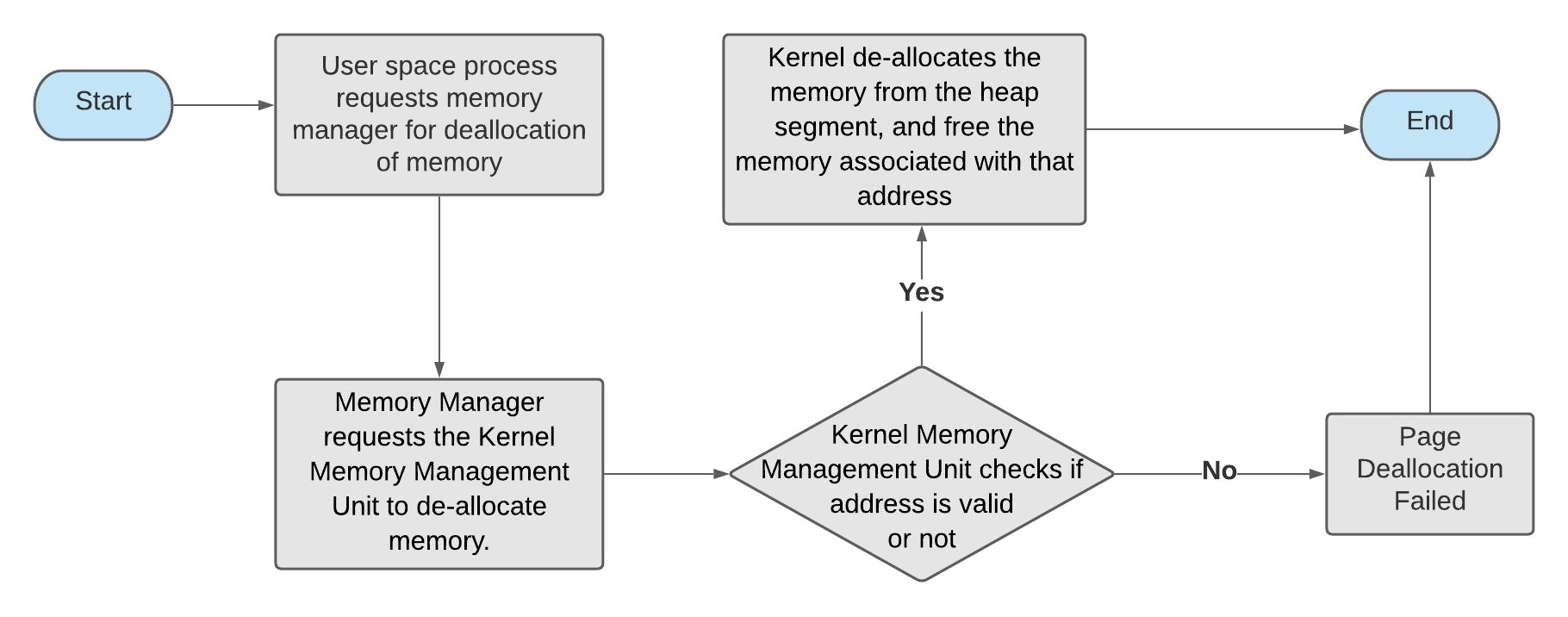


Fig 4: Algorithm for de-allocation of memory

2.2 Data/ Data structure

* Doubly Linked List
* Structures and Enumerators

2.3 SWOT Analysis

* Strength – Our memory manager shows statistics of the memory in use by every structure. Hence it becomes very easy to catch memory leaks. Our proposed techniques also address the problem of memory fragmentation.
* Weakness – If the user is running a larger number of processes on their system, then there is a possibility that the kernel memory management unit may not find a free virtual memory page and prompt error “page allocation failed” due to lack of memory.
* Opportunities - Data structures can grow and shrink according to the requirement. We can allocate (create) additional storage whenever we need them. We can de-allocate (free/delete) dynamic space whenever we are done with them. Dynamic Allocation is done at run time.
* Threats – There are basically two threats in our project. First, if Kernel finds no free memory in the heap segment, then it will not allocate memory to the user space process. Second, memory allocated by xmalloc() is not freed automatically which leads to garbage data collection hence it's important to free the allocated memory at the end of the program via xfree() function.

2.4 Project Features

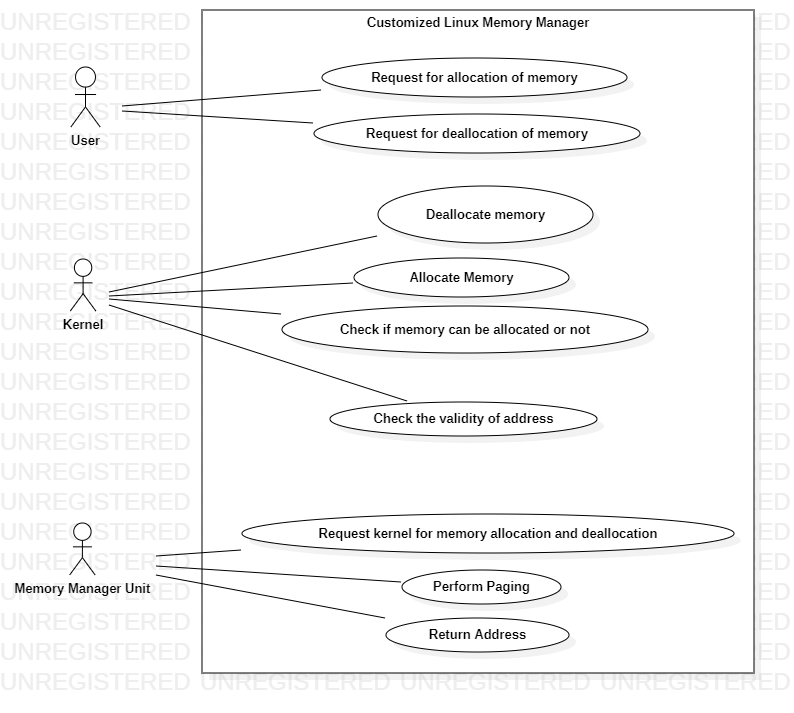
* Allocate and free the memory
* Catch memory leaks
* Show memory currently in use for every structure.
* Addressing memory fragmentation problems
* Demonstrates the internal design and implementation of malloc/free.

2.5 Design and Implementation Constraints

* There should be free memory for allocation through kernel memory management unit.

2.6 Design diagram

Fig 5 shows the use cases and the actors in the proposed Linux memory manager. User, Kernel and Memory Manager Unit are the actors of the system.

Fig 5. Use Case Diagram

3. SYSTEM REQUIREMENTS

3.1 User Interface

* We will develop a graphical user interface to show the statistics of memory in use via each structure.

3.2 Software Interface

* Kali Linux
* GCC Complier
* Gedit Editor

3.3 Database Interface

* No database is used in our project.

3.4 Protocols

* No protocols have been used

4. NON-FUNCTIONAL REQUIREMENTS

4.1 Performance requirements

* The proposed mechanism should be able to interface with different operating systems.
* The proposed technique should be accurate.
* The proposed technique should be better than the existing system.

4.2 Security requirements

* There are no specific security requirements for our proposed technique.

4.3 Software Quality Attributes

* The proposed technique should not crash under any circumstances.

5. OTHER REQUIREMENTS

* There are no other requirements for our proposed technique.

APPENDIX A: GLOSSARY

* Dynamic memory allocation - Dynamic memory allocation allows C programmers to allocate memory at runtime. Dynamic Memory Allocation is the manual allotment and releasing of memory based on your programming requirements. Dynamic memory is managed and served using pointers that point to newly allocated memory space in a region known as the heap.
* Memory Fragmentation - Memory fragmentation occurs when the majority of your memory is allocated in a large number of non-contiguous blocks, or chunks, leaving a significant portion of your total memory unassigned yet useless in most normal cases. As a result, out of memory exceptions or allocation issues occur (i.e. malloc returns null).
* Memory Leak - Memory leaks occur when programmers generate heap memory and then fail to destroy it. Memory leaks have the effect of lowering computer performance by decreasing the quantity of accessible memory.