**Error! Unknown document property name.**

**Error! Unknown document property name.**

High Level Design & Low Level Design

The purpose of this document is to provide with a template for documenting both HLD & LLD.

**Document Control :**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Project Revision History** | | | | | | | | |
|  |  |  | |  |  |  |  |  |
| **Date** | **Version** | **Author** | **Brief Description of Changes** | | | | **Approver Signature** | |
| 08-01-2023 | 0.1 | 06-01-2023\_LINUX17NOVB1\_SPRINT-2\_B | Introduction | | | |  | |
| 09-01-2023 | 0.1 | 06-01-2023\_LINUX17NOVB1\_SPRINT-2\_B | Project Scope | | | |  | |
| 10-01-2023 | 0.1 | 06-01-2023\_LINUX17NOVB1\_SPRINT-2\_B | Risk | | | |  | |
| 11-01-2023 | 0.1 | 06-01-2023\_LINUX17NOVB1\_SPRINT-2\_B | Design Overview | | | |  | |
| 12-01-2023 | 0.1 | 06-01-2023\_LINUX17NOVB1\_SPRINT-2\_B | Memory Management | | | |  | |
| 13-01-2023 | 0.1 | 06-01-2023\_LINUX17NOVB1\_SPRINT-2\_B | Performance | | | |  | |
| 14-01-2023 | 0.1 | 06-01-2023\_LINUX17NOVB1\_SPRINT-2\_B | Environment Description | | | |  | |
| 15-01-2023 | 0.1 | 06-01-2023\_LINUX17NOVB1\_SPRINT-2\_B | Mapping | | | |  | |
| 16-01-2023 | 0.1 | 06-01-2023\_LINUX17NOVB1\_SPRINT-2\_B | Reference | | | |  | |

[**1. Introduction**](#_heading=h.3znysh7) 5

[1.1. Intended Audience](#_heading=h.2et92p0) 5

[1.2. Acronyms/Abbreviations](#_heading=h.tyjcwt) 5

[1.3. Project Purpose](#_heading=h.3dy6vkm) 5

[1.4. Key Project Objectives](#_heading=h.1t3h5sf) 6

[1.5. Project Scope and Limitation](#_heading=h.4d34og8) 6

[*1.5.1. In Scope*](#_heading=h.2s8eyo1) 6

[*1.5.2. Out of scope*](#_heading=h.17dp8vu) 6

[1.6. Functional Overview](#_heading=h.3rdcrjn) 6

[1.7. Assumptions, Dependencies & Constraints](#_heading=h.26in1rg) 6

[1.8. Risks](#_heading=h.lnxbz9) 7

[**2. Design Overview**](#_heading=h.35nkun2) 7

[2.1. Design Objectives](#_heading=h.1ksv4uv) 7

[*2.1.1. Recommended Architecture*](#_heading=h.44sinio) 7

[2.2. Architectural Strategies](#_heading=h.2jxsxqh) 8

[*2.2.1. Design Alternative*](#_heading=h.z337ya) 8

[*2.2.2. Reuse of Existing Common Services/Utilities*](#_heading=h.3j2qqm3) 8

[*2.2.3. Creation of New Common Services/Utilities*](#_heading=h.1y810tw) 8

[*2.2.4. User Interface Paradigms*](#_heading=h.4i7ojhp) 9

[*2.2.5. System Interface Paradigms*](#_heading=h.2xcytpi) 9

[*2.2.6. Error Detection / Exceptional Handling*](#_heading=h.1ci93xb) 9

[*2.2.7. Memory Management*](#_heading=h.3whwml4) 9

[*2.2.8. Performance*](#_heading=h.2bn6wsx) 9

[*2.2.9. Security*](#_heading=h.qsh70q) 9

[*2.2.10. Concurrency and Synchronization*](#_heading=h.3as4poj) 10

[*2.2.11. Housekeeping and Maintenance*](#_heading=h.1pxezwc) 10

[**3. System Architecture**](#_heading=h.49x2ik5) 10

[3.1. System Architecture Diagram. (Not Necessary)](#_heading=h.2p2csry) 11

[3.2. System Use-Cases](#_heading=h.147n2zr) 12

[3.3. Subsystem Architecture](#_heading=h.3o7alnk) 12

[3.4. System Interfaces](#_heading=h.23ckvvd) 12

[*3.4.1. Internal Interfaces*](#_heading=h.ihv636) 12

[*3.4.2. External Interfaces*](#_heading=h.32hioqz) 13

[**4. Detailed System Design**](#_heading=h.1hmsyys) 13

[4.1. Key Entities](#_heading=h.41mghml) 13

[4.2. Detailed-Level Database Design](#_heading=h.2grqrue) 13

[*4.2.1. Data Mapping Information*](#_heading=h.vx1227) 14

[*4.2.2. Data Conversion*](#_heading=h.3fwokq0) 15

[4.3. Archival and retention requirements](#_heading=h.1v1yuxt) 15

[4.4. Disaster and Failure Recovery](#_heading=h.4f1mdlm) 15

[4.5. Business Process workflow](#_heading=h.2u6wntf) 15

[4.6. Business Process Modeling and Management (as applicable)](#_heading=h.19c6y18) 15

[4.7. Business Logic](#_heading=h.3tbugp1) 15

[4.8. Variables](#_heading=h.28h4qwu) 15

[4.9. Activity / Class Diagrams (as applicable)](#_heading=h.nmf14n) 15

[4.10. Data Migration](#_heading=h.37m2jsg) 15

[*4.10.1. Architectural Representation*](#_heading=h.1mrcu09) 15

[*4.10.2. Architectural Goals and Constraints*](#_heading=h.46r0co2) 15

[*4.10.3. Logical View*](#_heading=h.2lwamvv) 15

[*4.10.4. Architecturally Significant Design Packages*](#_heading=h.111kx3o) 15

[*4.10.5. Data model*](#_heading=h.3l18frh) 16

[*4.10.6. Deployment View*](#_heading=h.1egqt2p) 16

[**5. Environment Description**](#_heading=h.3ygebqi) 16

[5.1. Time Zone Support](#_heading=h.2dlolyb) 16

[5.2. Language Support](#_heading=h.sqyw64) 16

[5.3. User Desktop Requirements](#_heading=h.3cqmetx) 16

[5.4. Server-Side Requirements](#_heading=h.1rvwp1q) 16

[*5.4.1. Deployment Considerations*](#_heading=h.4bvk7pj) 16

[*5.4.2. Application Server Disk Space*](#_heading=h.2r0uhxc) 16

[*5.4.3. Database Server Disk Space*](#_heading=h.1664s55) 17

[*5.4.4. Integration Requirements*](#_heading=h.3q5sasy) 17

[*5.4.5. Jobs*](#_heading=h.25b2l0r) 17

[*5.4.6. Network*](#_heading=h.kgcv8k) 17

[*5.4.7. Others*](#_heading=h.34g0dwd) 17

[5.5. Configuration](#_heading=h.1jlao46) 17

[*5.5.1. Operating System*](#_heading=h.43ky6rz) 17

[*5.5.2. Database*](#_heading=h.2iq8gzs) 17

[*5.5.3. Network*](#_heading=h.xvir7l) 17

[*5.5.4. Desktop*](#_heading=h.3hv69ve) 18

[**6. References**](#_heading=h.1x0gk37) 18

[**7. Appendix**](#_heading=h.4h042r0) 18

# 

# Introduction

The buddy algorithm maintains free lists of different sized blocks. When a request for memory is made these free lists are searched. If the appropriate size is not found a larger block is split (variations of this algorithm determine how the block is split, for example, in a binary buddy system the block is split by powers of two).

## Intended Audience

|  |  |
| --- | --- |
| Anyone who want to use memory management using binary buddy algorithm |  |
|  |  |

## Acronyms/Abbreviations

|  |  |
| --- | --- |
| NONE |  |
|  |  |
|  |  |
|  |  |

## Project Purpose

The **binary** **buddy system** is a memory allocation and management algorithm that manages memory in **power of two increments**. Assume the memory size is 2U, suppose a size of S is required.

* **If 2U-1<S<=2U:** Allocate the whole block
* **Else:** Recursively divide the block equally and test the condition at each time, when it satisfies, allocate the block and get out the loop.
* System also keep the record of all the unallocated blocks each and can merge these different size blocks to make one big chunk.
* Consider a system having a buddy system with physical address space 128 KB. Calculate the size of partition for 18 KB process.
* So, the size of the partition for the 18 KB process = 32 KB. It divides by 2, till it is possible to get a minimum block to fit 18 KB.

## Key Project Objectives

Try to satisfy a memory request from user as suitably as possible

## Project Scope and Limitation

* Easy to implement a buddy system
* Allocates block of correct size
* It is easy to merge adjacent holes
* Fast to allocate memory and deallocate memory

Limitation

1. It requires all allocation unit to be powers of two.
2. It leads to internal fragmentation.

### In Scope

It allows users to allocate and deallocate memory quickly.

### Out of scope

System won’t allow users to put more than 32 memory blocks.

## Functional Overview

* Our project allows users to manage memory quickly.
* It gives users two options i.e., Allocation and deallocation.
* Program works on the equation 2^m, where m<=32.
* If users will put more than 2^32 memory the program will terminate.

## Assumptions, Dependencies & Constraints

* Related software or hardware
* Operating systems
* End-user characteristics
* Possible and/or probable changes in functionality.
* With the use of C language and Linux OS we can create this Binary Buddy project.
* This project will help the user to allocate and deallocate the memory easily and quickly.

## Risks

* Users must keep in mind that they cannot allocate memory more than 2 power 32 as that will lead to termination of the program.
* Users have to keep in mind that the request while using the binary buddy algorithm shows how much memory they want to allocate and how much they want to deallocate.

# Design Overview

The project is designed by dividing the program into multiple cpp files and creating the main function as a library and using that functionality on other test codes.

* These files contain multiple functions & declarations that are used for different cases. These functions are called based on the user input and his/her requirements.

## Design Objectives

Following are the operations performed in the file-

**-** Dynamic allocation of memory

**-** Deallocation of memory

**-** Display all the data with datetime stamp

### Recommended Architecture

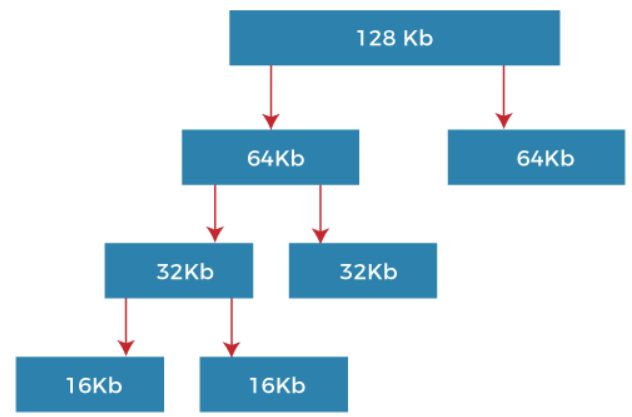
## Architectural Strategies

Architectural Strategies used in our project depends on the architecture of a binary buddy algorithm which breaks the whole chunk of memory into two equal parts.

* According to user request of memory allocate the nearest part of memory and
* For deallocation it will deallocate the memory as per user request and combine that free memory in a contiguous manner.

### Design Alternative

Example: Suppose the size of the memory segment is initially 256kb, and the kernel requests 25 kb of memory. The segment is initially divided into two buddies. Let's say A1 and A2, each 128 kb in size. One of these buddies is further divided into two 64kb buddies, B1 and B2. But the next highest power of 25 kb is 32kb so, either B1 or B2 is further divided into two 32kb buddies (C1 and C2), and finally, one of these buddies is used to satisfy the 25kb request. A split block can only be merged with its unique buddy block, which then reforms the larger block they were split from.



### Reuse of Existing Common Services/Utilities

In our project we are calling the log\_write() method to write all type of logs to log.txt

### Creation of New Common Services/Utilities

In our project we are creating a class config to read memory pool size from info.txt file. To start the algorithm user can configure the size of memory to be allocated for further requests.

### User Interface Paradigms

This service helps the users to easily visualize the memory management and then according to their requirements they can choose the available options*.*

### System Interface Paradigms

Good design is good business. If the system has interactive interface and it satisfies the user requirements then it's a plus point for developers .This Binary Buddy algorithm project can provide users with the appropriate services.

### Error Detection / Exceptional Handling

These elements are the main factors for cleanliness or to provide quality code.

**Maintainability**- Allows us to easily fix bugs without the fear of breaking current functionality.

**Extensibility**-Allows us to easily add to our code base, implementing new or changed requirements without breaking existing functionality.

**Readability**-Allows us to easily read code and discover its purpose without spending too much time digging.

### Memory Management

As our project is based on memory management, we have created different functions to allocate memory for users and deallocate the memory according to user request using Binary Buddy algorithm. Basically, this algorithm helps in memory management using buddy technique and we are using the same in our project.

### Performance

* Users can allocate the memory size according to their choice
* If the memory is of no use, then the user also has an option to deallocate the free memory for other requirements.

### Security

Not Applicable

### Concurrency and Synchronization

Not Applicable

### Housekeeping and Maintenance

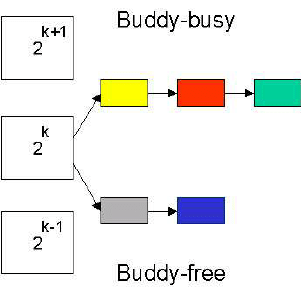
Not Applicable

# System Architecture

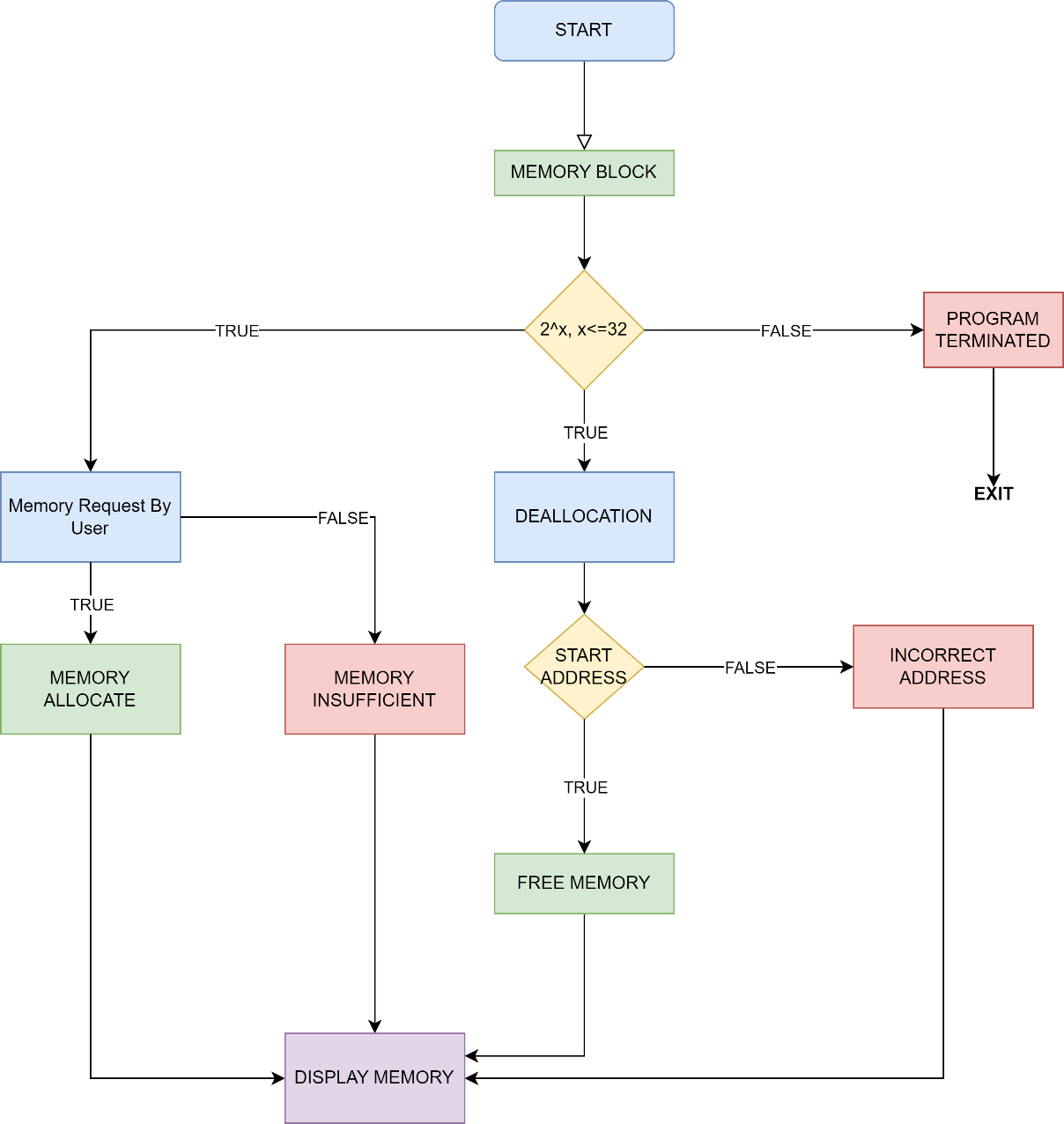
* The memory consists of a collection of blocks of consecutive memory, each of which is a power of two in size.
* Each block is marked either occupied or free, depending on whether it is allocated to the user.
* For each block we also know its size.
* The system provides two operations for supporting dynamic memory allocation.

1.Allocate(2^k): Finds a free block of size 2^k, marks it as occupied, and returns a pointer to it.

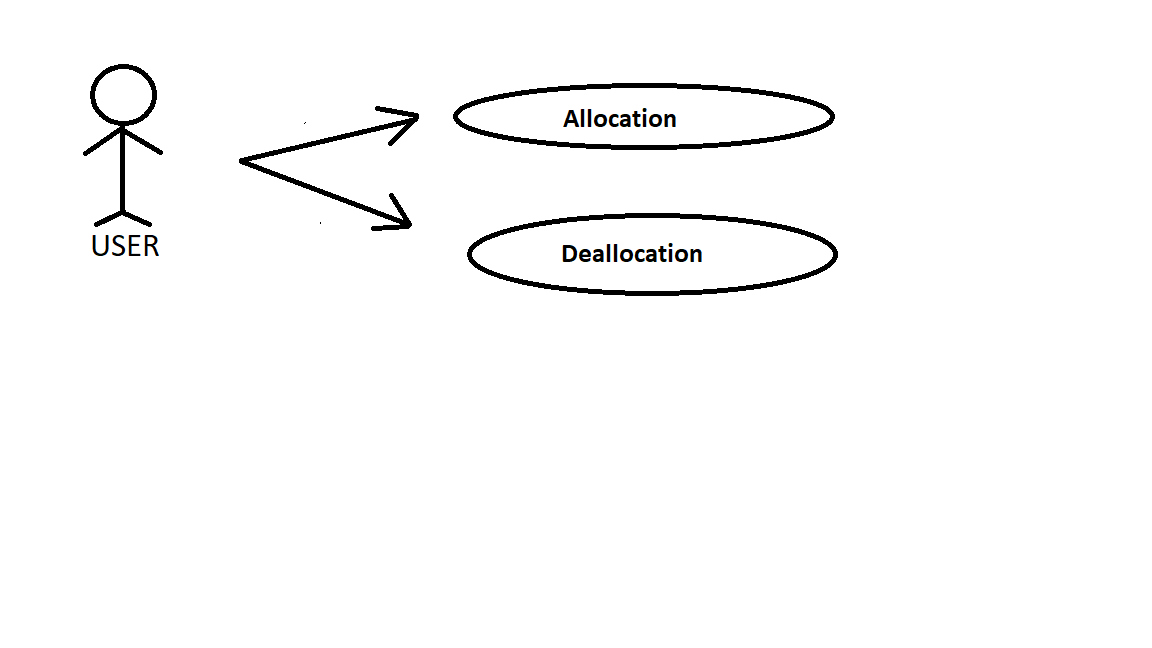
2.Deallocate(B): Marks the previously allocated block B as free and may merge it with others to form a larger free block.



## System architecture diagram



## System Use-Case



## Subsystem Architecture

Not Applicable

## System Interfaces

* The User interface depends upon the user.
* It depends upon the choice made by user according to his/her requirements.

### Internal Interfaces

As an Internal Interface we are using Ubuntu Linux distribution. It is an operating system that is made up of a collection of software based on Linux kernel or you can say distribution contains the Linux kernel and supporting libraries and software.

### External Interfaces

Not Applicable

# 

# Detailed System Design

**Static partition** schemes suffer from the **limitation** of having the fixed number of active processes and the usage of space may also not be optimal. The **buddy system** is a memory allocation and management algorithm that manages memory in **power of two increments**. Assume the memory size is 2U, suppose a size of S is required.

* **If 2U-1<S<=2U:** Allocate the whole block
* **Else:** Recursively divide the block equally and test the condition at each time, when it satisfies, allocate the block and get out the loop.
* System also keep the record of all the unallocated blocks each and can merge these different size blocks to make one big chunk.

## Key Entities

* Allocation
* Deallocation

## Detailed-Level Database Design

Not Applicable

### Data Mapping Information

**4.2.1.1 main()**

Here the user is provided with options to choose from like allocate, deallocate or exit according to user choice

**4.2.1.2. allocate()**

This function is used to allocate the memory according to user request so that the memory can be allocated.

**4.2.1.3 give\_fitter()**

This provide the end value so that the memory can be divided in two parts according to use request.

**4.2.1.4 give\_slot()**

This function is use to get the address of the memory, the start address and end address gives the size of the memory being allocated

**4.2.1.5 deallocate()**

This function is used to deallocate the block of memory so that the next user can make the request to allocate the memory.

**4.2.1.6 buddies()**

This function is used to divide a block of memory into two parts according to the size of the memory request made by the user.

**4.2.1.7 check\_merge()**

This function is used to merge the block of memory together so that we can save the block of memory that is not in use.

**4.2.1.7.1 is\_power\_of\_two()**

This function is used to check if a number is whole power of two or not. This is used to validate the user input, whether its correct or not.

**4.2.1.8**  **display()**

This function is used to display the starting and ending address and display it on terminal and show the errors that occur when a wrong request is made. This function gives a single block to print\_node function to print a single node at a time on the screen.

**4.2.1.9**  **print\_node()**

This function takes a single block of memory as argument and prints its information on the console. Like starting Address of memory block, ending address , size of the block , total space acquired and whether its been allocated or not.

**4.2.1.10**  **get\_Pool\_Size()**

This function returns the memory pool size by reading it from the config file named **“info.txt”** present in the project folder only. If no **“info.txt”** file is there it will by default return 1024KB of memory.

**4.2.1.11 get\_Log\_Path()**

This function returns the path of the file to which all the logs are appended. It reads the file path from the variable named log\_Path in config file named**“info.txt”.**

**4.2.1.12 get\_timestamp()**

This function return the current timestamp at the runtime of code. It is used for logging purpose only to log every request’s timestamp in the logfile.

It uses pre-defined chrono and time library.

**4.2.1.13 write\_log()**

This function takes message and request type as arguments and logs them to the log file.

Along with the message and request type also prints the timestamp with every request. It uses file handling.

### Data Conversion

## Archival and retention requirements

Not Applicable

## Disaster and Failure Recovery

Not Applicable

## Business Process workflow

Not Applicable

## Business Process Modeling and Management (as applicable)

Not Applicable

## Business Logic

Not Applicable

## Variables

Not Applicable

## Activity / Class Diagrams (as applicable)

Not Applicable

## Data Migration

Not Applicable

### Architectural Representation

Not Applicable

### Architectural Goals and Constraints

Not Applicable

### Logical View

Not Applicable

### Architecturally Significant Design Packages

Not Applicable

### Data model

Not Applicable

### Deployment View

Not Applicable

# Environment Description

Not Applicable

## Time Zone Support

The number of seconds of time difference between the local time zone and Coordinated Universal Time [UTC].

## Language Support

C++ language is used in this project.

## User Desktop Requirements

* Windows: 7 or above
* Linux

## Server-Side Requirements

Not Applicable

### Deployment Considerations

Not Applicable

### Application Server Disk Space

Not Applicable

### Database Server Disk Space

Not Applicable

### Integration Requirements

Not Applicable

### Jobs

Not Applicable

### Network

Not Applicable

### Others

Not Applicable

## Configuration

For Binary Buddy Algorithm we need Windows 7 or above and we have also used Linux.

### Operating System

* Windows: 7 or above
* Linux

### Database

Not Applicable

### Network

Not Applicable

### Desktop

Ubuntu (LINUX)

# References

* <https://www.geeksforgeeks.org/operating-system-allocating-kernel-memory-buddy-system-slab-system/>
* https://www.cs.fsu.edu/~engelen/courses/COP402003/p827.pdf

# Appendix

* LINUX commands
* C++ Functions

**Change Log**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **QMS Template Version Control (Maintained by QA)** | | | | | |
|  |  |  |  |  |  |
| **Date** | **Version** | **Author** | | **Description** | |
| 08-April-2023 | 1.0 | QA Team | | Initial Version | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |