



Heap Memory Manager

By: Mohamed Refat

Provided to : Eng Reda Mahher

TABLE OF CONTENTS

01

Overview

03

Testting

02

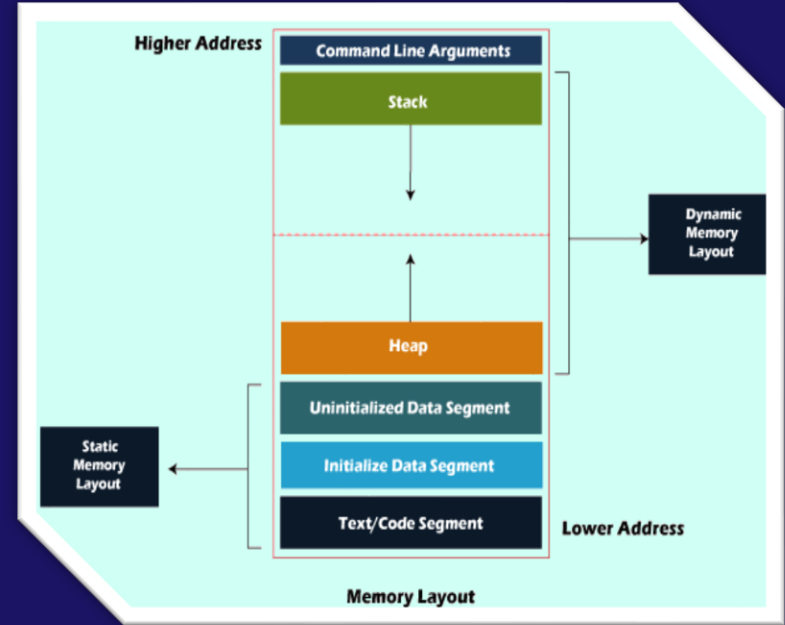
Implementation

04

Pros and
cons

INTRODUCTION

Heap management is an essential aspect of programming in the C language, particularly in Linux environments. The heap is a region of memory used for dynamic memory allocation, allowing programs to request and release memory at runtime. Heap management facilitates efficient memory usage and enables the creation of dynamic data structures, such as linked lists, trees, and arrays.

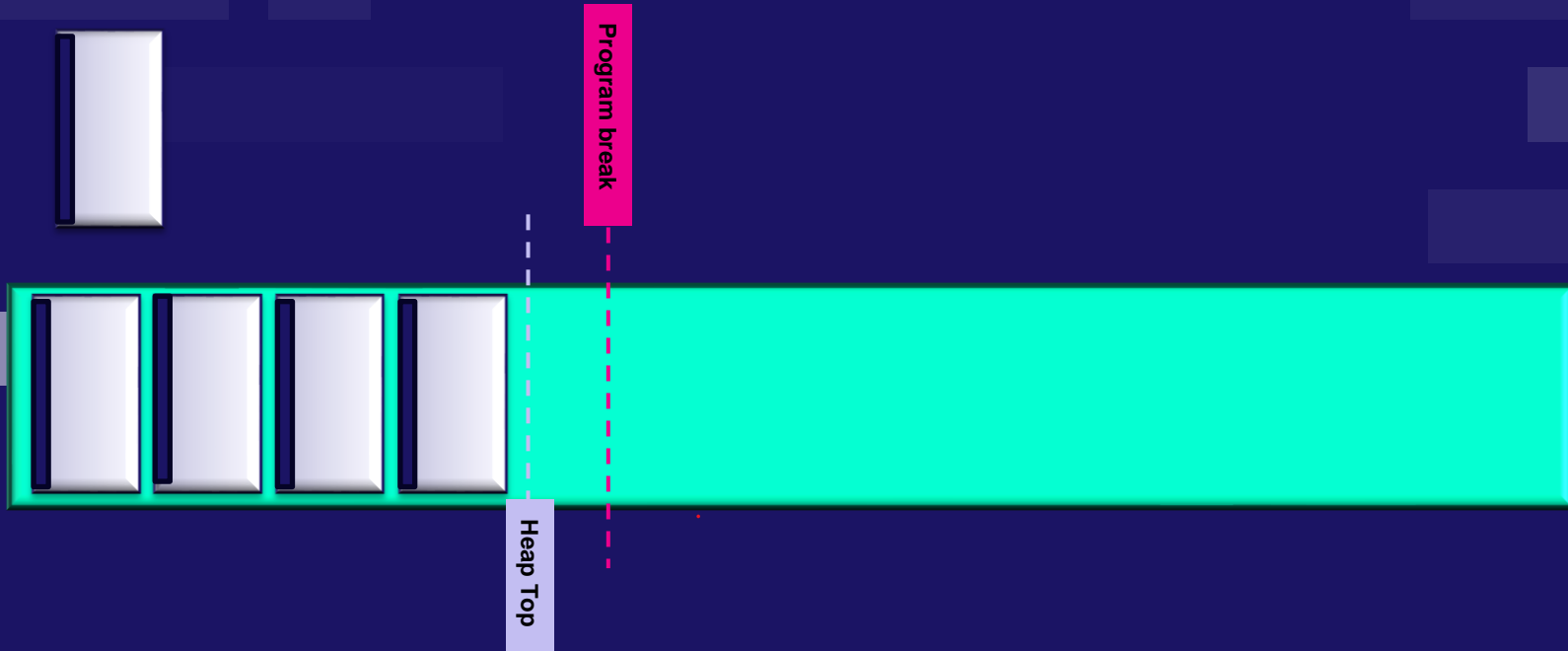




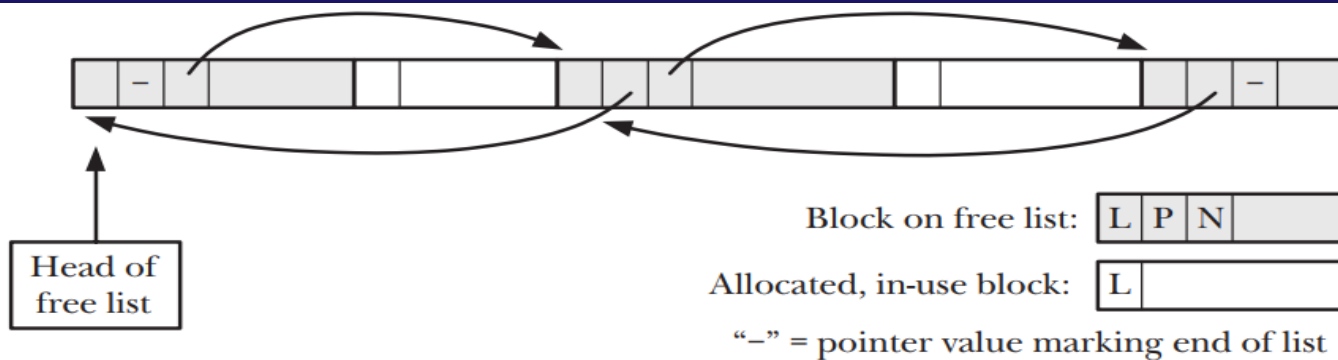
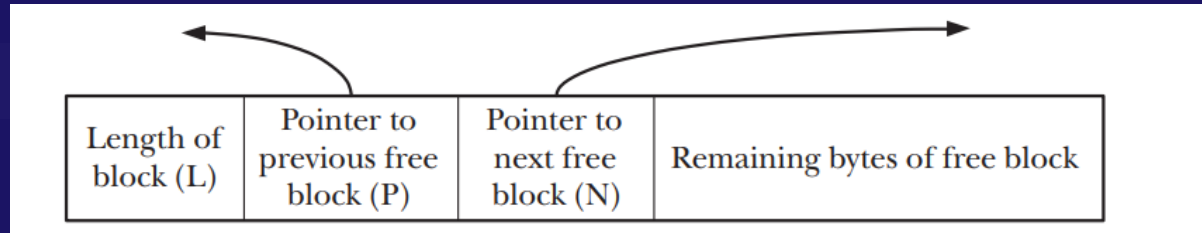
01

OVERVIEW

Allocation And Deallocation



Allocation And Deallocation





02

Project Implementation

Main interfaces



Malloc

malloc is a function in C that dynamically allocates memory during runtime.

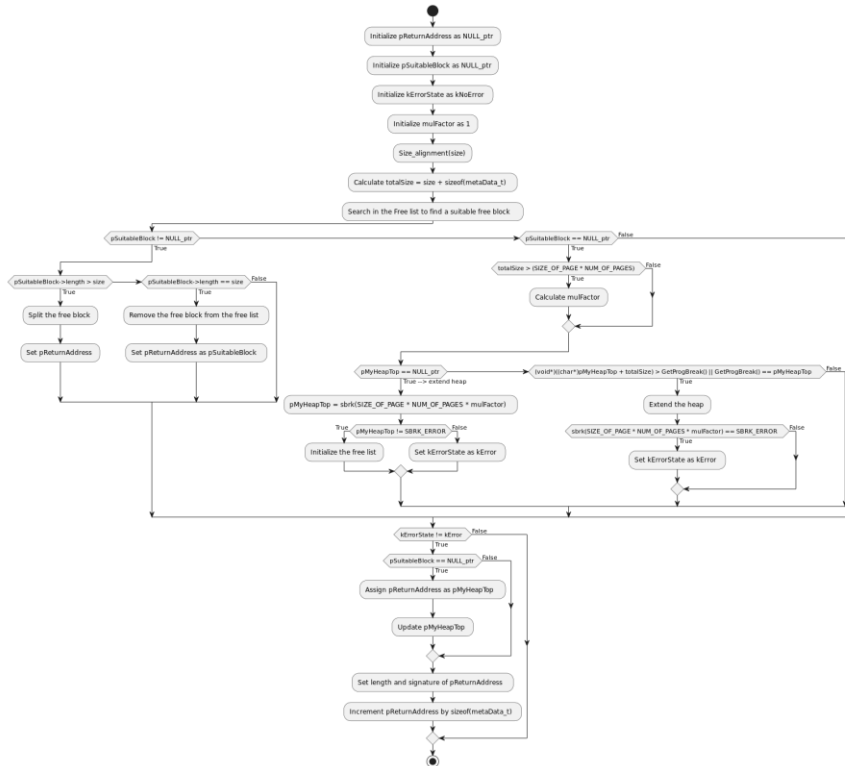
1. Search for a suitable free block in the free list using the `FreeList_FindSuitableBlock()` function, based on either the first-fit or best-fit technique.

2. If there is no suitable free block in the free list:

- Allocate from the heap top if there is unallocated space
- if not extend the heap by calling `sbrk()`




Malloc() Flow Chart



Free

The free function in C is used to deallocate memory previously allocated with malloc or calloc

- 
1. Read the metadata
 2. Check if the pointer is a valid pointer by verifying the signature in the metadata
 3. Insert the block into the free list using the `FreeList_InsertBlock()`
 4. merging of adjacent free blocks in order to reduce fragmentation and optimize memory usage
 5. Lowering the program break by releasing unused memory is important for efficient memory management, preventing memory leaks, optimizing address space.



Free() Flow Chart

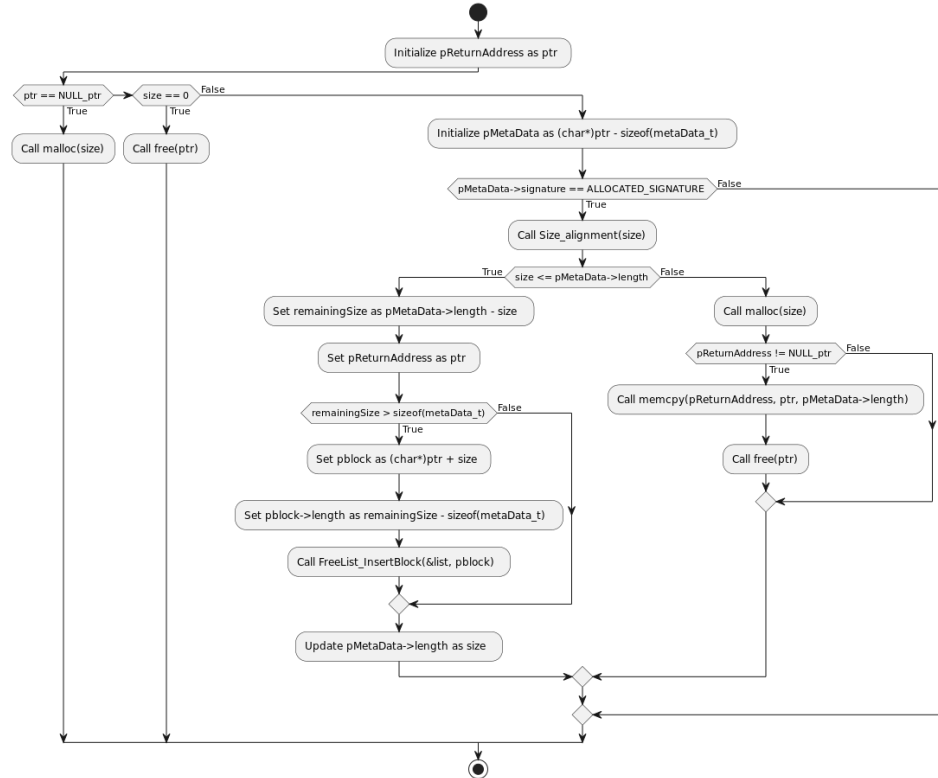




realloc

`Realloc()`: is a function in C used to dynamically resize a previously allocated block of memory. It allows you to increase or decrease the size of the memory block, preserving the existing data if possible.

realloc() Flow Chart

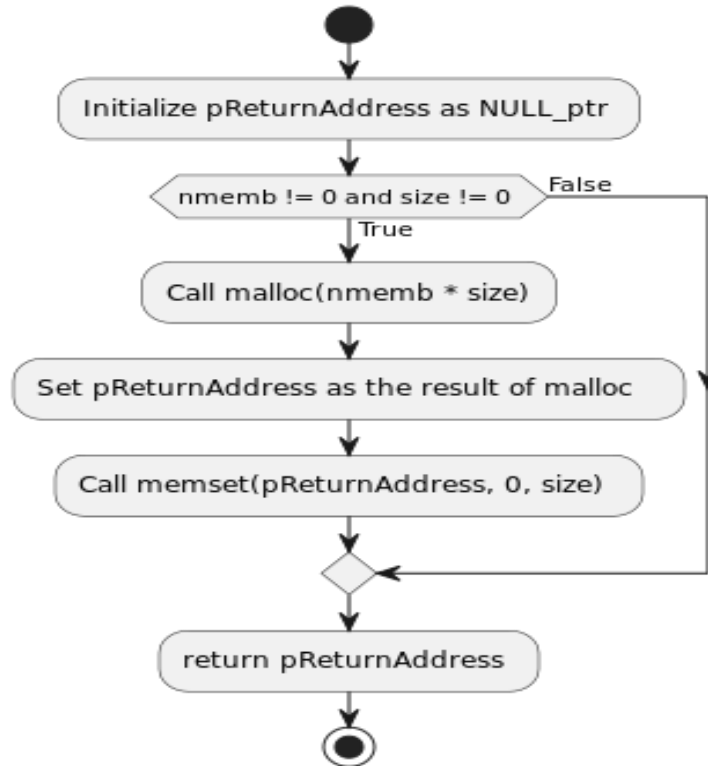




Calloc

`Calloc()`: is a function in C used to dynamically allocate and initialize memory for multiple elements, typically in an array format. It takes two parameters: the number of elements to allocate and the size of each element. The `calloc` function calculates the total amount of memory required based on the provided parameters and initializes all the bytes to zero

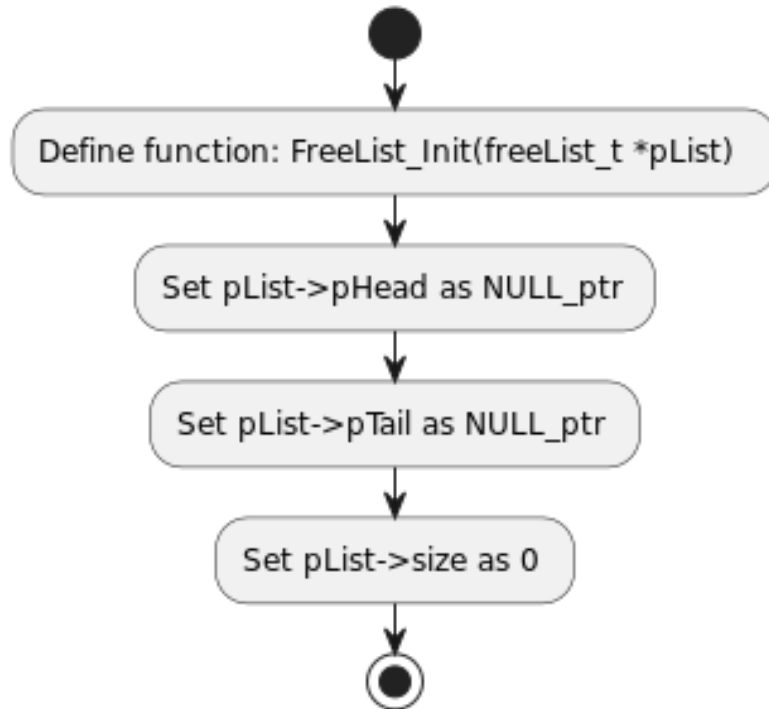
calloc() Flow Chart



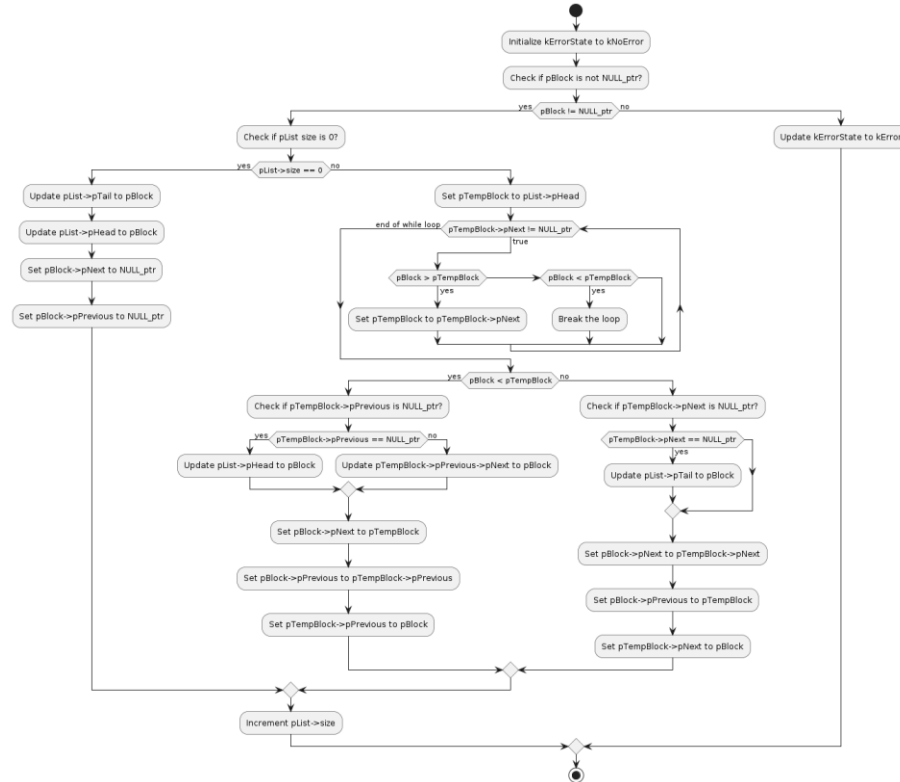
Free List interfaces



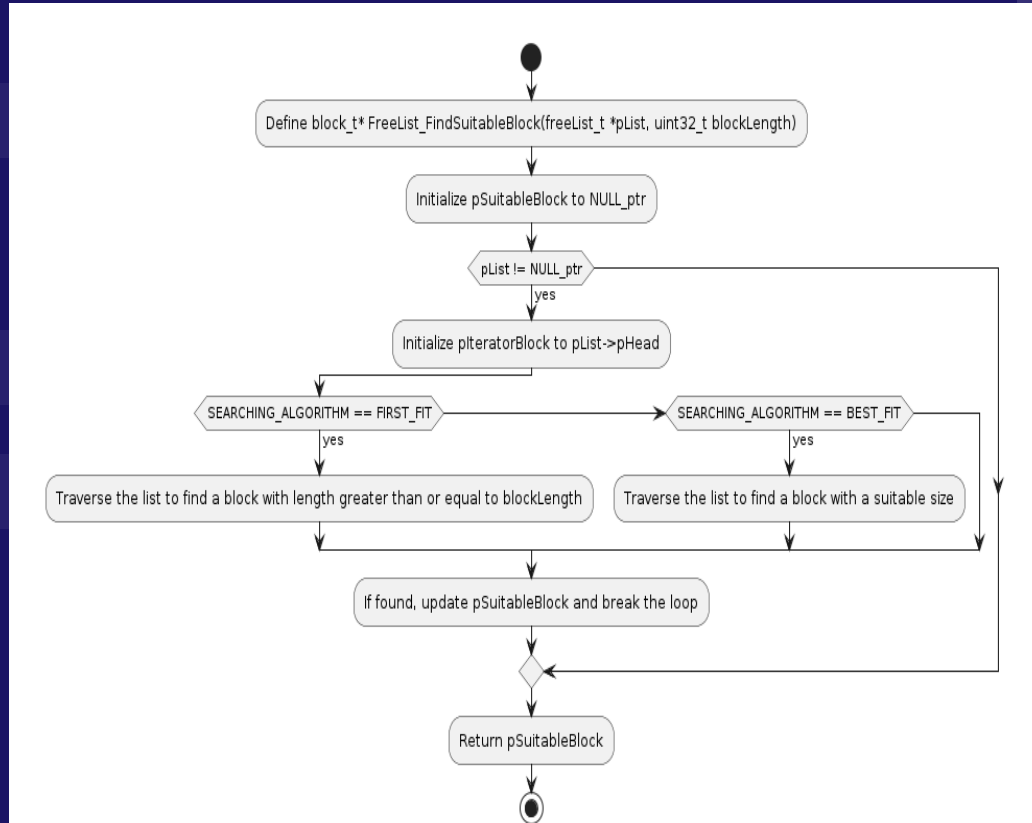
FreeList Init() Flow Chart



FreeList_InsertBlock()Flow Chart

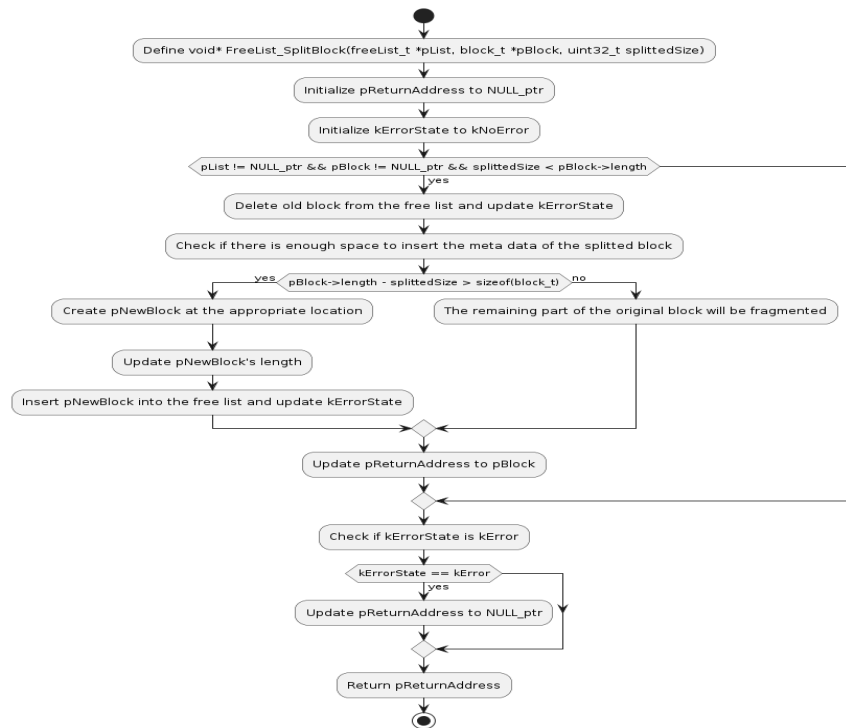


FreeList_FindSuitableBlock()Flow Chart



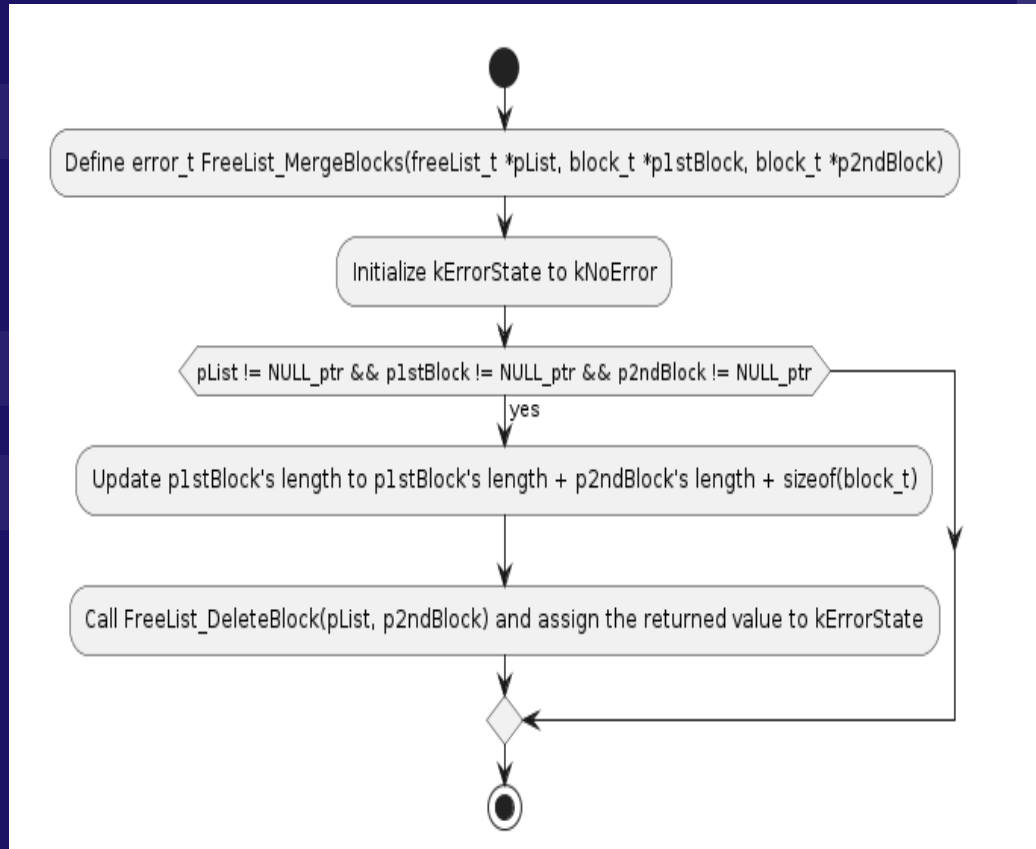


FreeList SplitBlock() Flow Chart

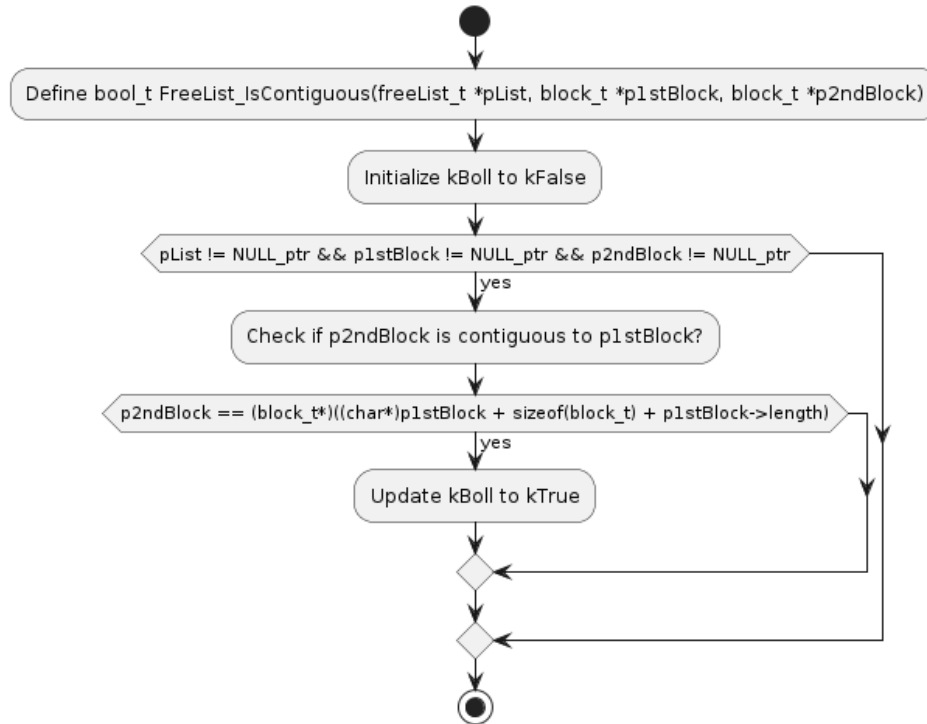




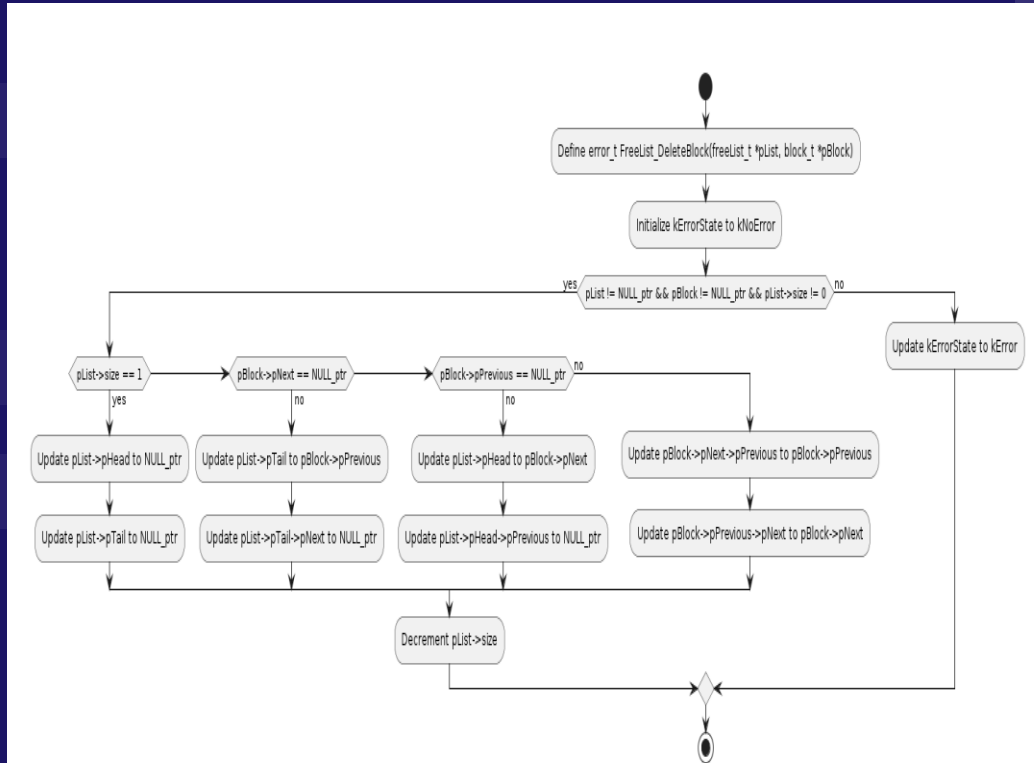
FreeList MergeBlocks() Flow Chart



FreeList_IsContiguous() Flow Chart



FreeList DeleteBlock() Flow Chart

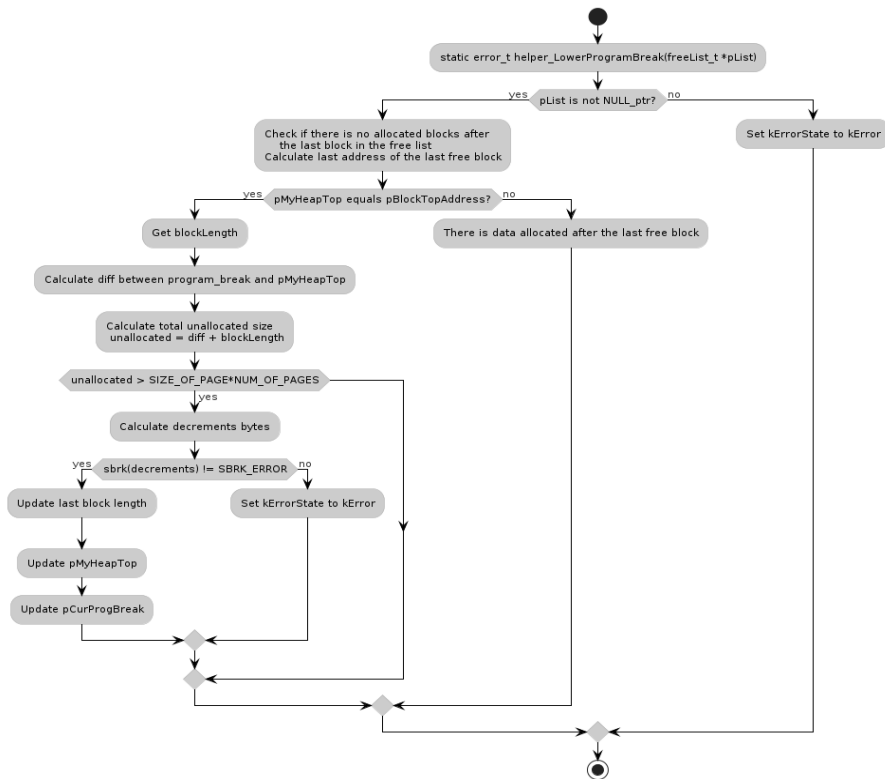


Helper interfaces





helper_LowerProgramBreak() Flow Chart





03

Testing

free_and_sbrk()

```
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$ ./a.exe 10000 1000 1 5000 10000
etext = 0x5a80a42e2115, edata=0x5a80a42e5010, end=0x5a80a42e5048, initial program break=0x5a80a5d1e000
Initial program break:      0x5a80a5d3f000
Allocating [numAllocs]: 10000 * [blockSize]1000 bytes
Before free(), program break is: 0x5a80a66ea000
Freeing blocks from 5000 to 10000 in steps of 1
After free(), program break is: 0x5a80a6204000
Free List size is: 1
Last Block size is: 16336
```

Random free_and_sbrk()

```
Initial program break:          0x57a886312000
Allocating [numAllocs]: 1847 * [blockSize]6659 bytes
Before free(), program break is: 0x57a886ecd000
Freeing blocks from 678 to 1315 in steps of 1
After free(), program break is: 0x57a886ecd000
Free List size is: 1
Last Block size is: 4266920
-----
Running program iteration 97
etext = 0x59980ee0c115, edata=0x59980ee0f010, end=0x59980ee0f048, initial program break=0x59980f082000
Initial program break:          0x59980f0a3000
Allocating [numAllocs]: 23508 * [blockSize]21883 bytes
Before free(), program break is: 0x59982dbc5000
Freeing blocks from 1371 to 7111 in steps of 1
After free(), program break is: 0x59982dbc5000
Free List size is: 1
Last Block size is: 125796768
-----
Running program iteration 98
etext = 0x5daee8171115, edata=0x5daee8174010, end=0x5daee8174048, initial program break=0x5daee9089000
Initial program break:          0x5daee90aa000
Allocating [numAllocs]: 10472 * [blockSize]10018 bytes
Before free(), program break is: 0x5daee4f4000
Freeing blocks from 235 to 10085 in steps of 1
After free(), program break is: 0x5daee4f4000
Free List size is: 1
Last Block size is: 98982824
-----
Running program iteration 99
etext = 0x653b93979115, edata=0x653b9397c010, end=0x653b9397c048, initial program break=0x653b9434a000
Initial program break:          0x653b9436b000
Allocating [numAllocs]: 14949 * [blockSize]7389 bytes
Before free(), program break is: 0x653b9ad1f000
Freeing blocks from 14913 to 14917 in steps of 1
After free(), program break is: 0x653b9ad1f000
Free List size is: 1
Last Block size is: 37056
-----
Running program iteration 100
etext = 0x636f10810115, edata=0x636f10813010, end=0x636f10813048, initial program break=0x636f1081d000
Initial program break:          0x636f1083e000
Allocating [numAllocs]: 1874 * [blockSize]11477 bytes
Before free(), program break is: 0x636f11cbd000
Freeing blocks from 825 to 1077 in steps of 1
After free(), program break is: 0x636f11cbd000
Free List size is: 1
Last Block size is: 2910488
```

Random allocation and free

```
Freeing remaining memory at address 0x117659c0
Freeing remaining memory at address 0xaaafebc0
Freeing remaining memory at address 0x13aa5470
Freeing remaining memory at address 0xccec64d0
Freeing remaining memory at address 0x132082f0
Freeing remaining memory at address 0x11f376d0
Freeing remaining memory at address 0xd852570
Freeing remaining memory at address 0x10e00120
Freeing remaining memory at address 0xd054ad0
Freeing remaining memory at address 0x4fd9510
Freeing remaining memory at address 0x133b0a40
Freeing remaining memory at address 0x10110420
Freeing remaining memory at address 0xc1862a0
Freeing remaining memory at address 0x88ef8a0
Freeing remaining memory at address 0xc3d6970
Freeing remaining memory at address 0x52b01c0
Freeing remaining memory at address 0x15757690
Freeing remaining memory at address 0x5e98040
Test complete.
==8050==
==8050== HEAP SUMMARY:
==8050==   in use at exit: 0 bytes in 0 blocks
==8050==   total heap usage: 502,516 allocs, 502,516 frees, 25,713,766,563 bytes allocated
==8050==
==8050== All heap blocks were freed -- no leaks are possible
==8050==
==8050== For lists of detected and suppressed errors, rerun with: -s
==8050== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

real    0m26.737s
user    0m17.296s
sys     0m5.734s
mohamedrefat@mohamedrefat-VirtualBox: ~/Desktop/HMM/HMM$
```

Simple Linux utilities: LS,cat..etc

```
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$  
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$  
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$ export LD_PRELOAD=/home/mohamedrefat/Desktop/HMM/HMM/libhmm.so  
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$ ls  
FreeList.c          FreeList.h  HMM_Config.h  HMM.h        Makefile      README.md     test-debug  
FreeList.dynamic.o  HMM.c      HMM.dynamic.o libhmm.so    random-test.sh STD_TYPES.h   Tests  
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$ cat ./test-debug/script.sh  
make clean  
sudo rm -rf /var/crash  
sudo rm -rf ./vim_dump  
ulimit -c unlimited  
sudo systemctl restart apport  
gcc -c -g -fPIC *.c  
gcc -shared -o libhmm.so *.o  
LD_PRELOAD=/home/mohamedrefat/Desktop/HMM/HMM/libhmm.so vim  
apport-unpack /var/crash/_usr_bin_vim.basic.1000.crash ./vim_dump  
gdb vim ./vim_dump/CoreDump  
  
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$
```

Linux utilities: bash

```
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$  
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$  
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$ export LD_PRELOAD=/home/mohamedrefat/Desktop/HMM/HMM/libhmm.so  
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$ ls  
FreeList.c      FreeList.h  HMM_Config.h  HMM.h        Makefile      README.md     test-debug  
FreeList.dynamic.o  HMM.c      HMM.dynamic.o  libhmm.so    random-test.sh  STD_TYPES.h   Tests  
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$ cat ./test-debug/script.sh  
make clean  
sudo rm -rf /var/crash  
sudo rm -rf ./vim_dump  
ulimit -c unlimited  
sudo systemctl restart apport  
gcc -c -g -fPIC *.c  
gcc -shared -o libhmm.so *.o  
LD_PRELOAD=/home/mohamedrefat/Desktop/HMM/HMM/libhmm.so vim  
apport-unpack /var/crash/_usr_bin_vim.basic.1000.crash ./vim_dump  
gdb vim ./vim_dump/CoreDump  
  
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$
```




04

pros and cons

Pros and Cons



Performance

- Optimized design
- Minimum call of system calls (`sbrk()`)



Abstraction

- well-defined interfaces, enhancing modularity and flexibility



Saving Memory

- Lower Program break
- Almost no fragmentation



**Not all
utilitues wokrs**

Vim doesn't work

THANKS!

Do you have any questions?

