Heap Memory Manager

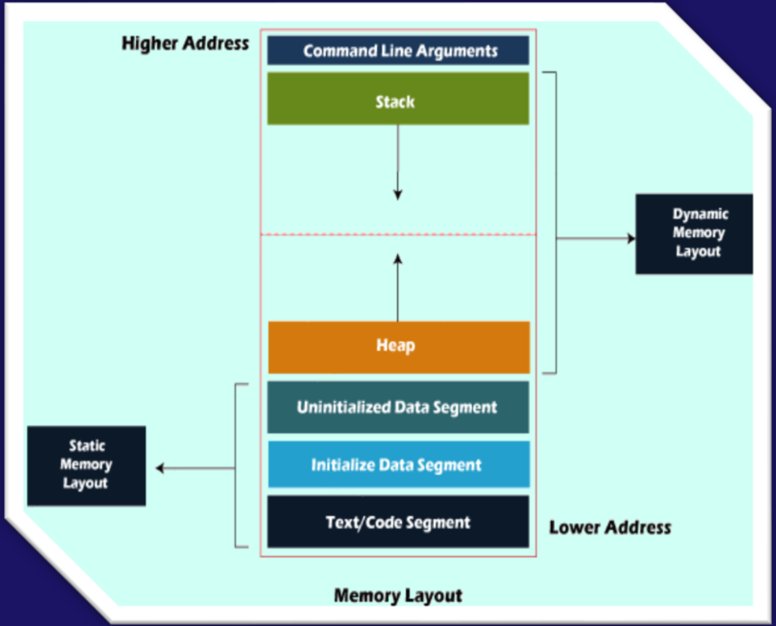
By: Mohamed Refat

Provided to : Eng Reda Mahher

TABLE OF CONTENTS

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.
- Key goals include dynamic memory allocation, deallocation, implementing various memory management policies, optimizing performance, and integrating error handling mechanisms.
- This project offers essential heap functionalities such as malloc, calloc, realloc, and free



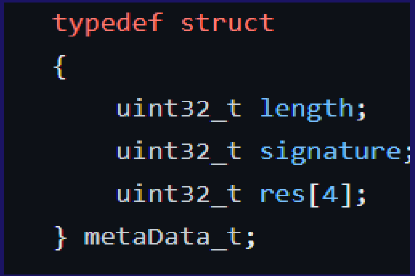
01

OVERVIEW

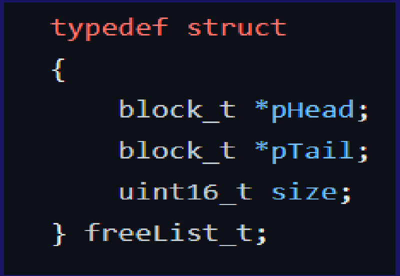
Project Structure

Allocated block

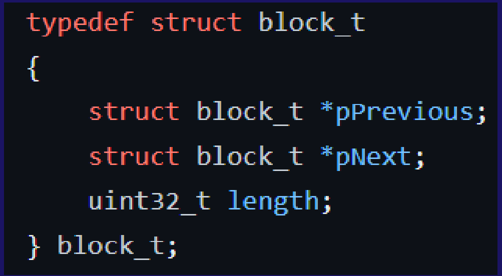
Freed block



```c
typedef struct
{
    uint32_t length;
    uint32_t signature;
    uint32_t res[4];
} metaData_t;
```

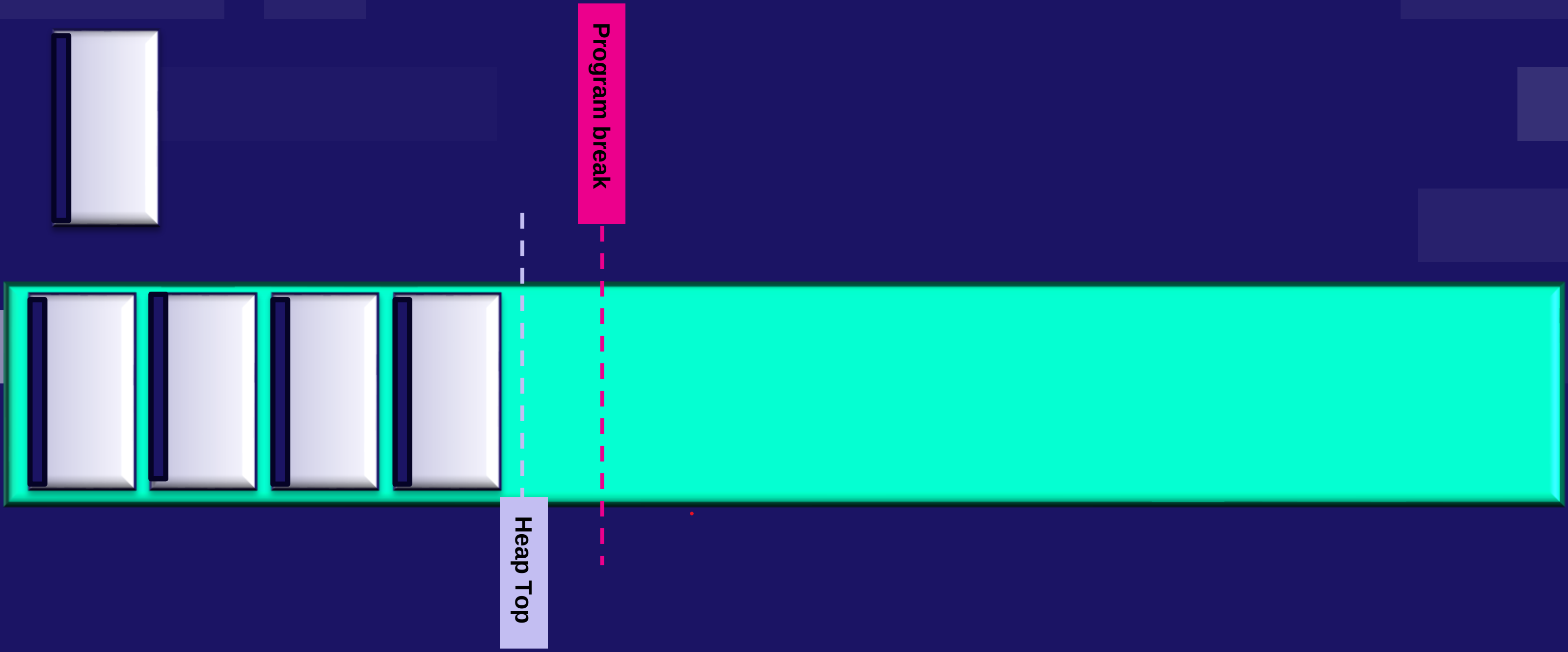


```c
typedef struct
{
    block_t *pHead;
    block_t *pTail;
    uint16_t size;
} freeList_t;
```

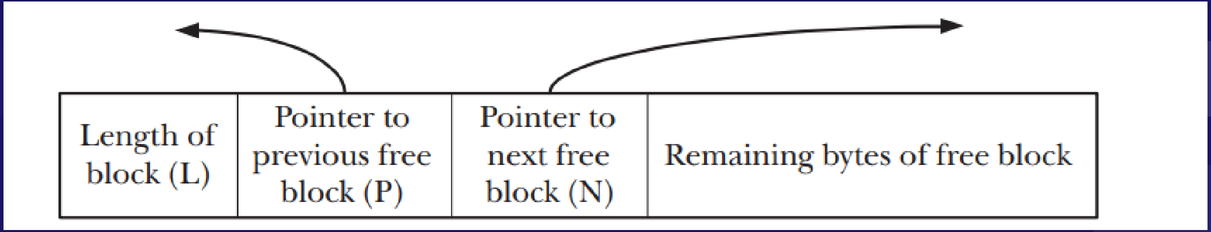


```c
typedef struct block_t
{
    struct block_t *pPrevious;
    struct block_t *pNext;
    uint32_t length;
} block_t;
```

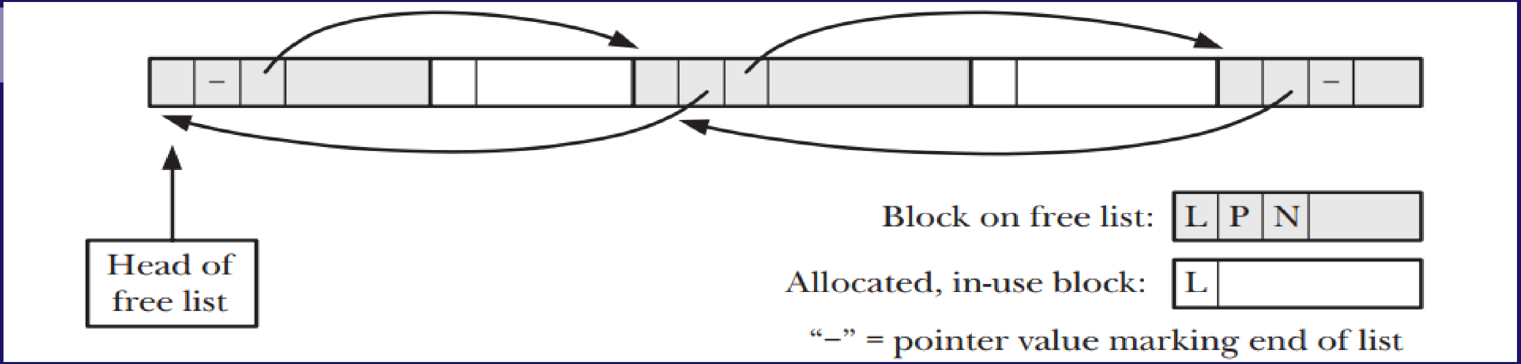
Allocation And Deallocation



Allocation And Deallocation



Length of block (L)
Pointer to previous free block (P)
Pointer to next free block (N)
Remaining bytes of free block



02

Project Implementation

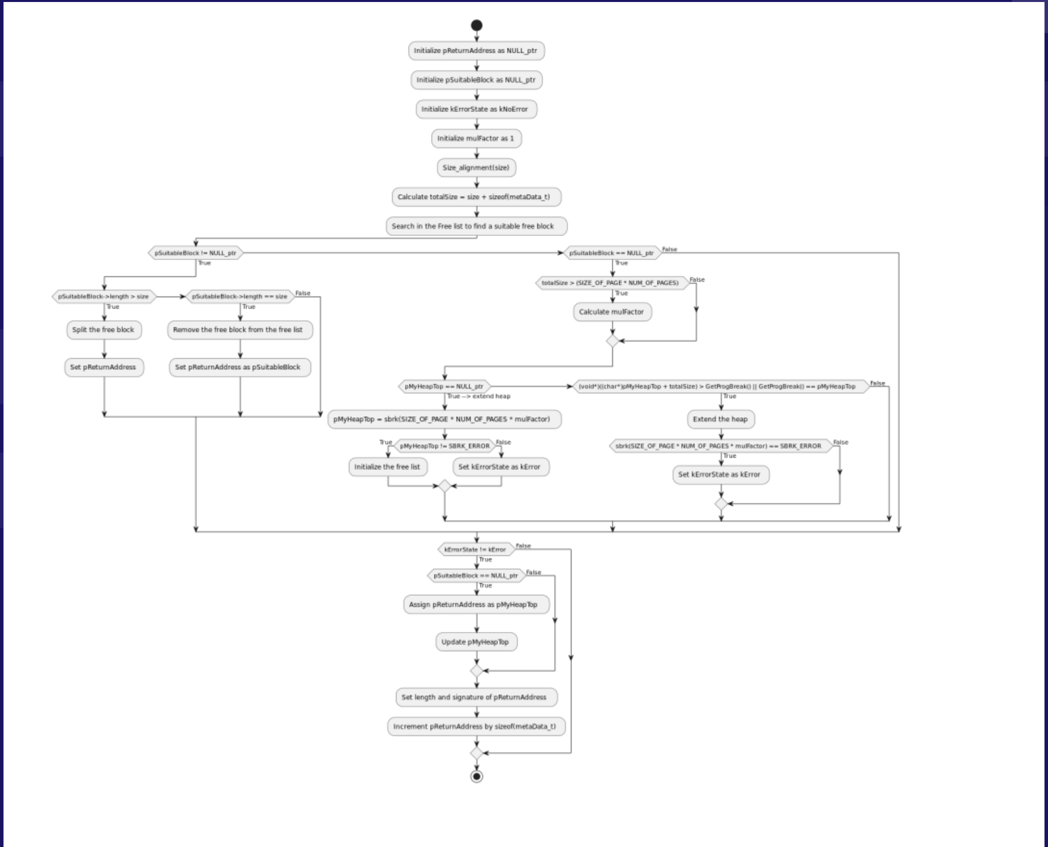
Main interfaces

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

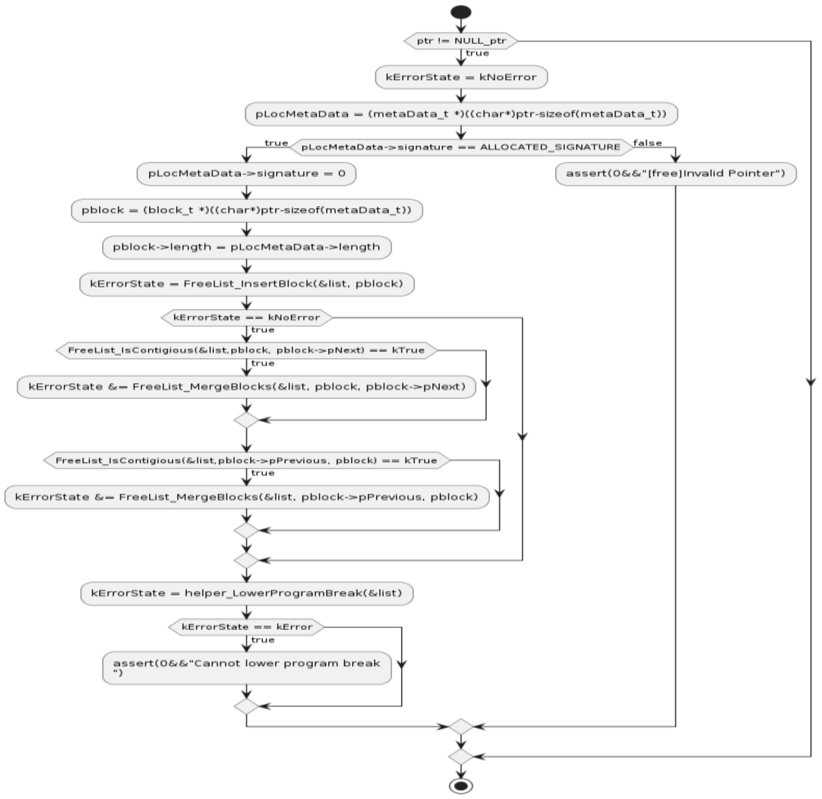

Initialize pReturnAddress as NULL_ptr
Initialize pSuitableBlock as NULL_ptr
Initialize kErrorState as kNoError
Initialize mulFactor as 1
Size_alignment(size)
Calculate totalSize = size + sizeof(metaData_t)
Search in the Free list to find a suitable free block
pSuitableBlock != NULL_ptr
True
pSuitableBlock->length > size
True
Split the free block
Set pReturnAddress
pSuitableBlock->length == size
True
False
Remove the free block from the free list
Set pReturnAddress as pSuitableBlock
pSuitableBlock == NULL_ptr
False
True
totalSize > (SIZE_OF_PAGE * NUM_OF_PAGES)
False
True
Calculate mulFactor
pMyHeapTop == NULL_ptr
True --> extend heap
pMyHeapTop = sbrk(SIZE_OF_PAGE * NUM_OF_PAGES * mulFactor)
True
pMyHeapTop != SBRK_ERROR
False
Initialize the free list
Set kErrorState as kError
(void*)((char*)pMyHeapTop + totalSize) > GetProgBreak() || GetProgBreak() == pMyHeapTop
False
True
Extend the heap
sbrk(SIZE_OF_PAGE * NUM_OF_PAGES * mulFactor) == SBRK_ERROR
False
True
Set kErrorState as kError
kErrorState != kError
False
True
pSuitableBlock == NULL_ptr
False
True
Assign pReturnAddress as pMyHeapTop
Update pMyHeapTop
Set length and signature of pReturnAddress
Increment pReturnAddress by sizeof(metaData_t)

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.

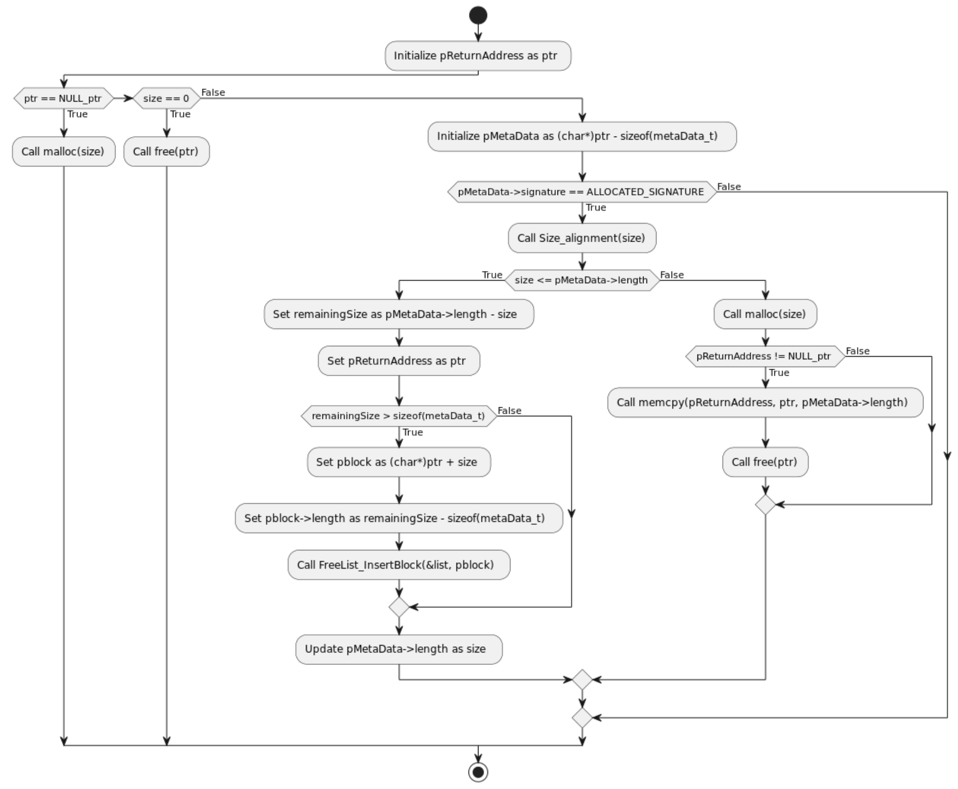
Free() Flow Chart


ptr != NULL_ptr
true
kErrorState = kNoError
pLocMetaData = (metaData_t *)((char*)ptr-sizeof(metaData_t))
true
pLocMetaData->signature == ALLOCATED_SIGNATURE
false
pLocMetaData->signature = 0
assert(0&&"[free]Invalid Pointer")
pblock = (block_t *)((char*)ptr-sizeof(metaData_t))
pblock->length = pLocMetaData->length
kErrorState = FreeList_InsertBlock(&list, pblock)
kErrorState == kNoError
true
FreeList_IsContigious(&list,pblock, pblock->pNext) == kTrue
true
kErrorState &= FreeList_MergeBlocks(&list, pblock, pblock->pNext)
FreeList_IsContigious(&list,pblock->pPrevious, pblock) == kTrue
true
kErrorState &= FreeList_MergeBlocks(&list, pblock->pPrevious, pblock)
kErrorState = helper_LowerProgramBreak(&list)
kErrorState == kError
true
assert(0&&"Cannot lower program break ")

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

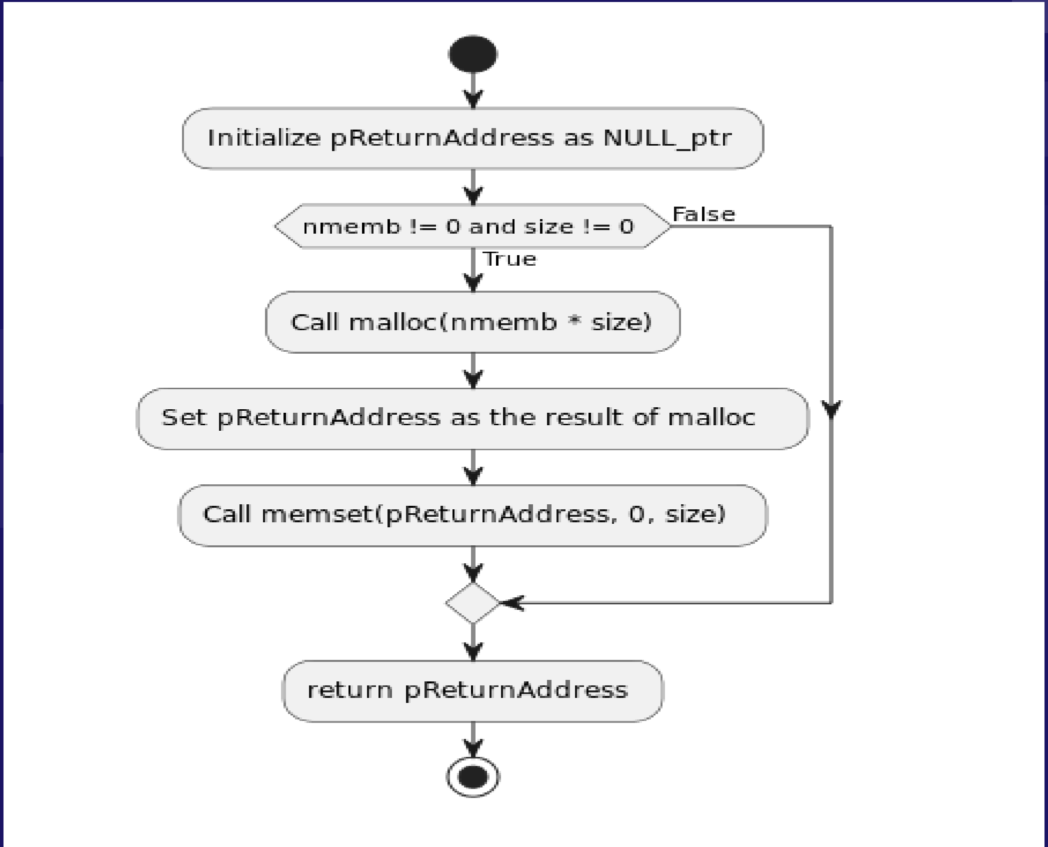


Initialize pReturnAddress as ptr
ptr == NULL_ptr
True
Call malloc(size)
size == 0
True
Call free(ptr)
False
Initialize pMetaData as (char*)ptr - sizeof(metaData_t)
pMetaData->signature == ALLOCATED_SIGNATURE
True
False
Call Size_alignment(size)
size <= pMetaData->length
True
False
Set remainingSize as pMetaData->length - size
Set pReturnAddress as ptr
remainingSize > sizeof(metaData_t)
True
False
Set pblock as (char*)ptr + size
Set pblock->length as remainingSize - sizeof(metaData_t)
Call FreeList_InsertBlock(&list, pblock)
Update pMetaData->length as size
Call malloc(size)
pReturnAddress != NULL_ptr
True
False
Call memcpy(pReturnAddress, ptr, pMetaData->length)
Call free(ptr)

Calloc

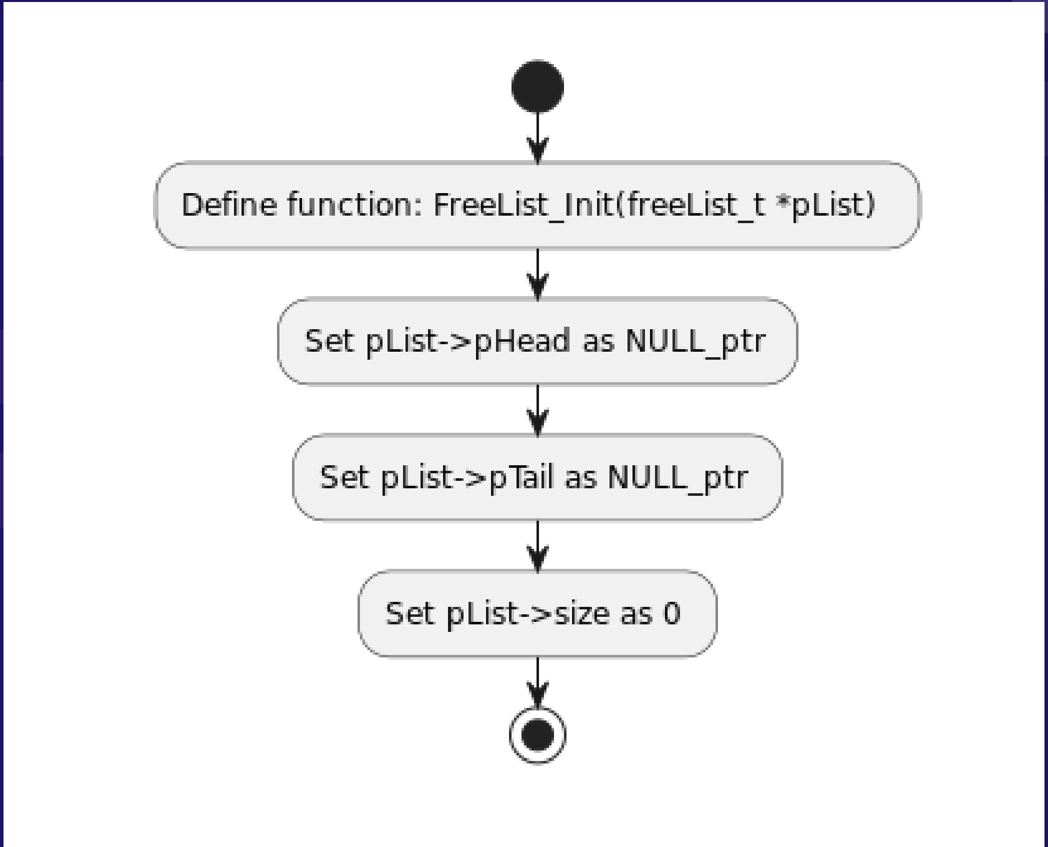
Calloc(): is a function in C used to dynamically allocate and initialize memory for multiple elements, typically in an array format. It takes two parameters: the number of elements to allocate and the size of each element. The calloc function calculates the total amount of memory required based on the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

🔗 calloc() Flow Chart

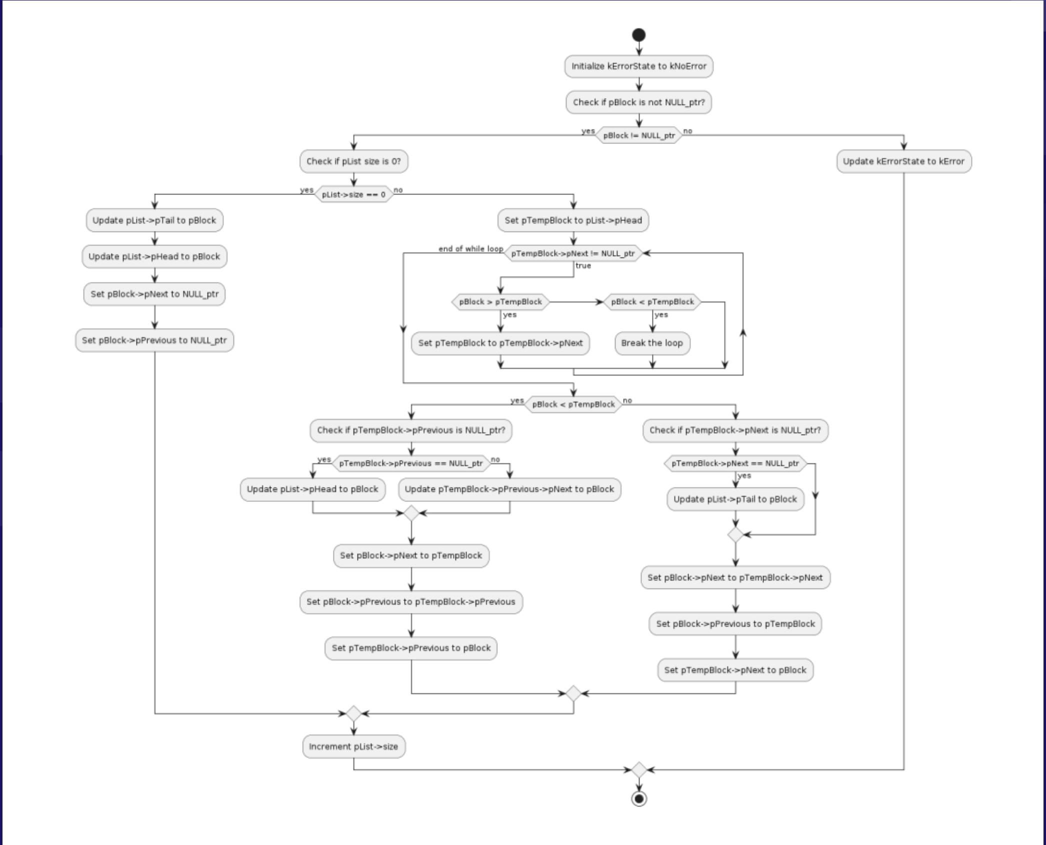

Initialize pReturnAddress as NULL_ptr
nmemb != 0 and size != 0
False
True
Call malloc(nmemb * size)
Set pReturnAddress as the result of malloc
Call memset(pReturnAddress, 0, size)
return pReturnAddress

Free List interfaces

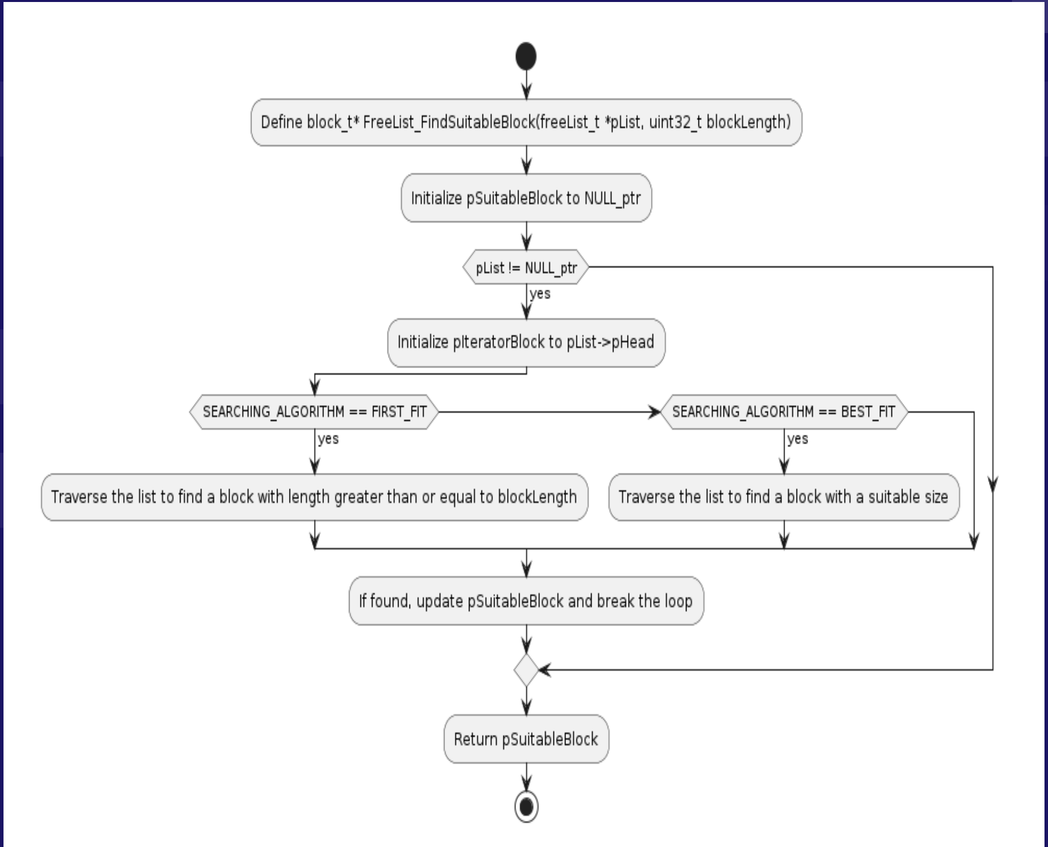
🔗FreeList_Init() Flow Chart


Define function: FreeList_Init(freeList_t *pList)
Set pList->pHead as NULL_ptr
Set pList->pTail as NULL_ptr
Set pList->size as 0

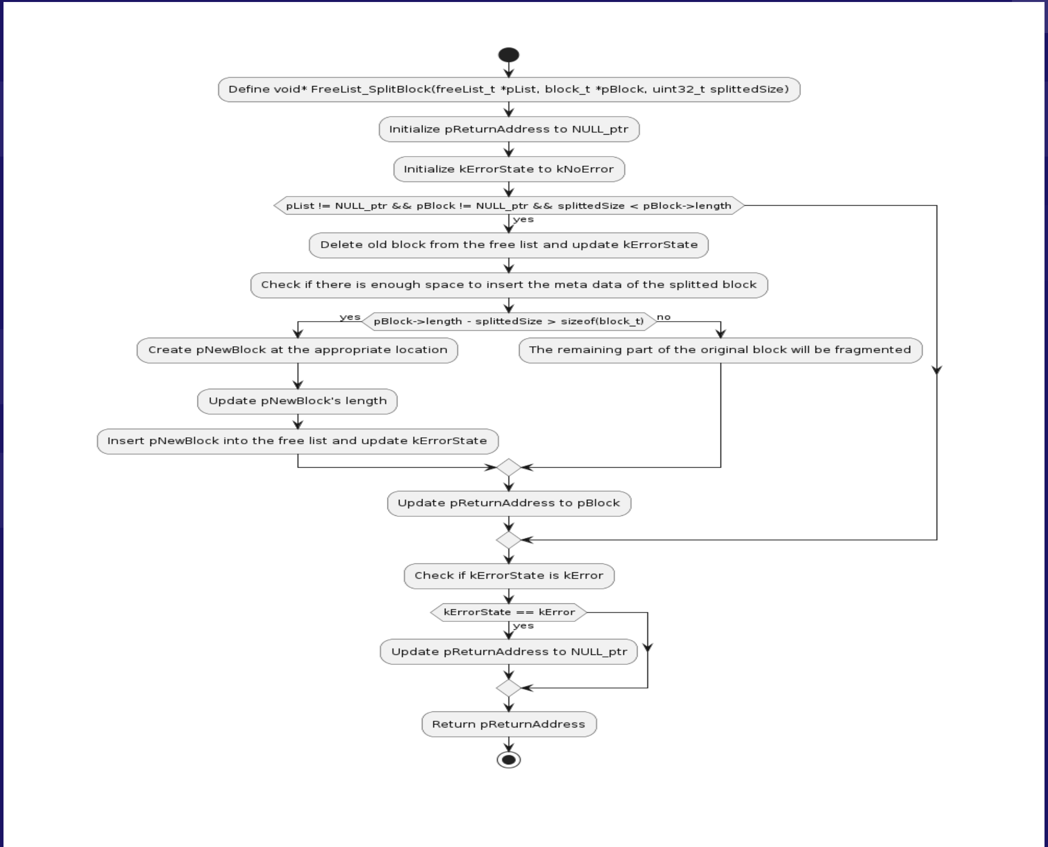
FreeList_InsertBlock()Flow Chart


Initialize kErrorState to kNoError
Check if pBlock is not NULL_ptr?
yes
pBlock != NULL_ptr
no
Check if pList size is 0?
Update kErrorState to kError
yes
pList->size == 0
no
Update pList->pTail to pBlock
Set pTempBlock to pList->pHead
Update pList->pHead to pBlock
end of while loop
pTempBlock->pNext != NULL_ptr
true
Set pBlock->pNext to NULL_ptr
pBlock > pTempBlock
pBlock < pTempBlock
yes
yes
Set pTempBlock to pTempBlock->pNext
Break the loop
Set pBlock->pPrevious to NULL_ptr
yes
pBlock < pTempBlock
no
Check if pTempBlock->pPrevious is NULL_ptr?
Check if pTempBlock->pNext is NULL_ptr?
yes
pTempBlock->pPrevious == NULL_ptr
no
pTempBlock->pNext == NULL_ptr
yes
Update pList->pHead to pBlock
Update pTempBlock->pPrevious->pNext to pBlock
Update pList->pTail to pBlock
Set pBlock->pNext to pTempBlock
Set pBlock->pNext to pTempBlock->pNext
Set pBlock->pPrevious to pTempBlock->pPrevious
Set pBlock->pPrevious to pTempBlock
Set pTempBlock->pPrevious to pBlock
Set pTempBlock->pNext to pBlock
Increment pList->size

FreeList_FindSuitableBlock()Flow Chart

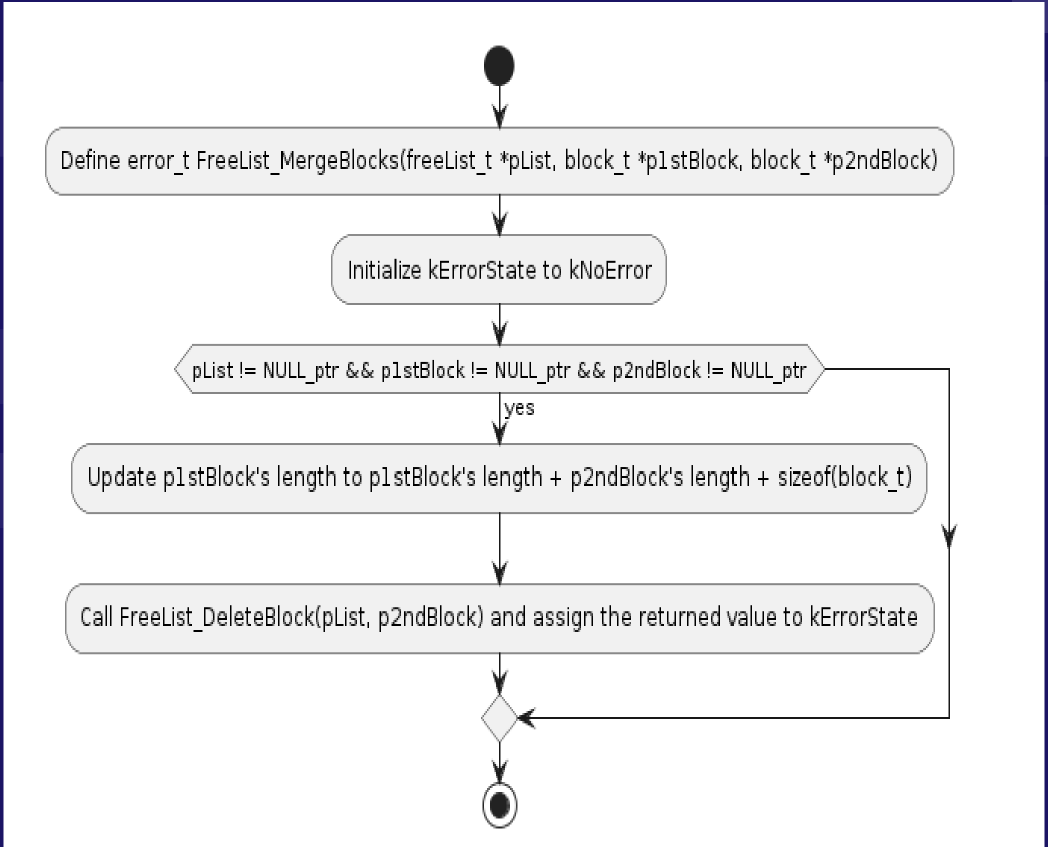

Define block_t* FreeList_FindSuitableBlock(freeList_t *pList, uint32_t blockLength)
Initialize pSuitableBlock to NULL_ptr
pList != NULL_ptr
yes
Initialize pIteratorBlock to pList->pHead
SEARCHING_ALGORITHM == FIRST_FIT
yes
Traverse the list to find a block with length greater than or equal to blockLength
SEARCHING_ALGORITHM == BEST_FIT
yes
Traverse the list to find a block with a suitable size
If found, update pSuitableBlock and break the loop
Return pSuitableBlock

FreeList_SplitBlock()Flow Chart



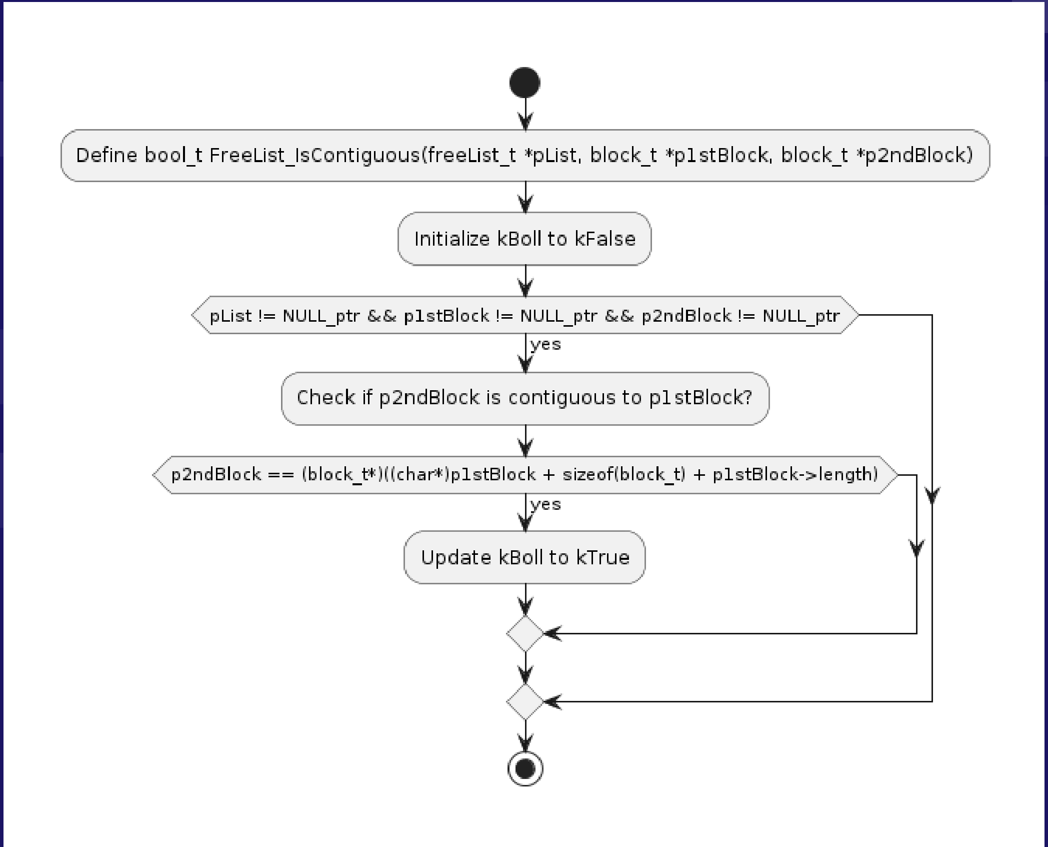
Define void* FreeList_SplitBlock(freeList_t *pList, block_t *pBlock, uint32_t splittedSize)
Initialize pReturnAddress to NULL_ptr
Initialize kErrorState to kNoError
pList != NULL_ptr && pBlock != NULL_ptr && splittedSize < pBlock->length
yes
Delete old block from the free list and update kErrorState
Check if there is enough space to insert the meta data of the splitted block
yes
pBlock->length - splittedSize > sizeof(block_t)
no
Create pNewBlock at the appropriate location
The remaining part of the original block will be fragmented
Update pNewBlock's length
Insert pNewBlock into the free list and update kErrorState
Update pReturnAddress to pBlock
Check if kErrorState is kError
kErrorState == kError
yes
Update pReturnAddress to NULL_ptr
Return pReturnAddress

🔗 FreeList_MergeBlocks() Flow Chart

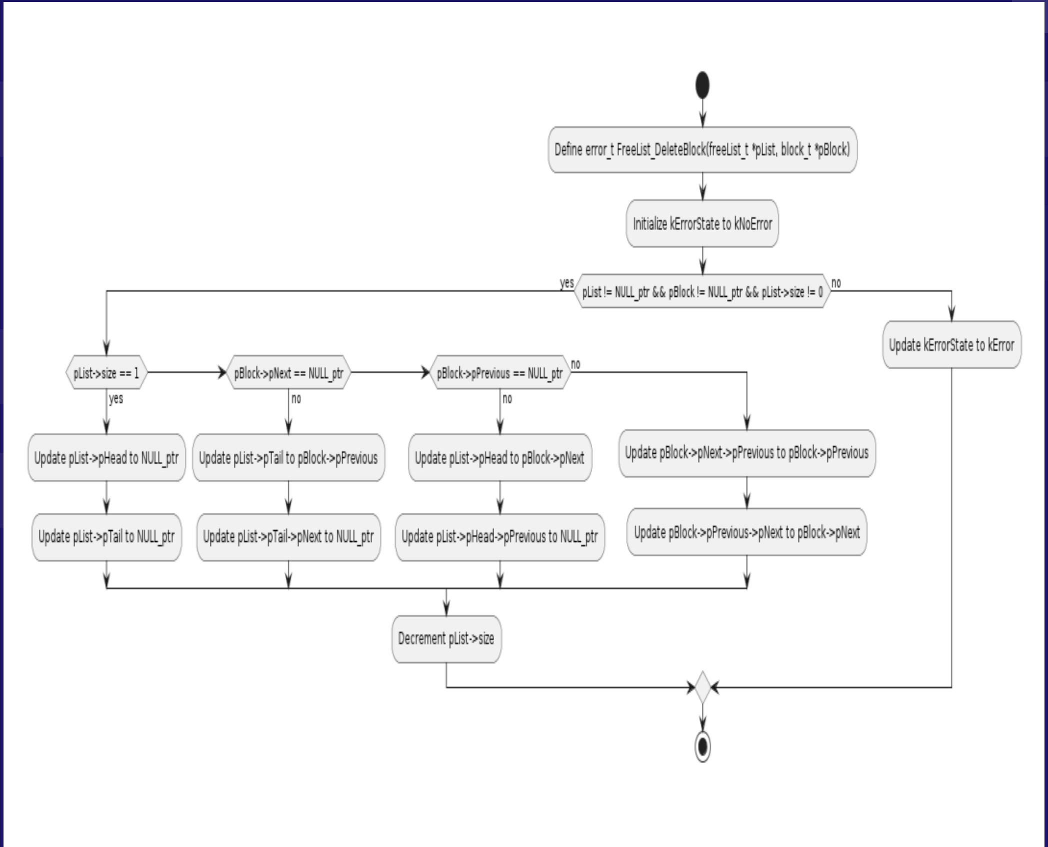


Define error_t FreeList_MergeBlocks(freeList_t *pList, block_t *p1stBlock, block_t *p2ndBlock)
Initialize kErrorState to kNoError
pList != NULL_ptr && p1stBlock != NULL_ptr && p2ndBlock != NULL_ptr
yes
Update p1stBlock's length to p1stBlock's length + p2ndBlock's length + sizeof(block_t)
Call FreeList_DeleteBlock(pList, p2ndBlock) and assign the returned value to kErrorState

🔗FreeList_IsContiguous()Flow Chart

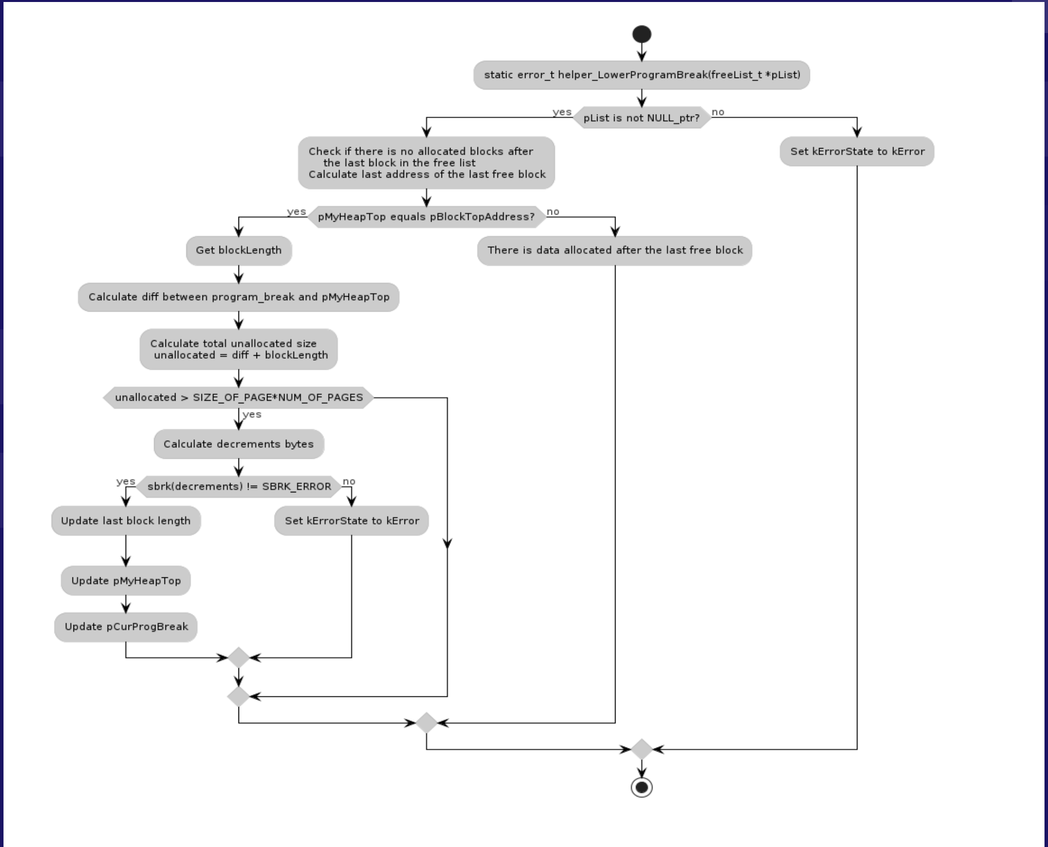

Define bool_t FreeList_IsContiguous(freeList_t *pList, block_t *p1stBlock, block_t *p2ndBlock)
Initialize kBoll to kFalse
pList != NULL_ptr && p1stBlock != NULL_ptr && p2ndBlock != NULL_ptr
yes
Check if p2ndBlock is contiguous to p1stBlock?
p2ndBlock == (block_t*)((char*)p1stBlock + sizeof(block_t) + p1stBlock->length)
yes
Update kBoll to kTrue

FreeList_DeleteBlock()Flow Chart


Define error_t FreeList_DeleteBlock(freeList_t *pList, block_t *pBlock)
Initialize kErrorState to kNoError
yes
pList != NULL_ptr && pBlock != NULL_ptr && pList->size != 0
no
Update kErrorState to kError
pList->size == 1
yes
pBlock->pNext == NULL_ptr
no
pBlock->pPrevious == NULL_ptr
no
no
Update pList->pHead to NULL_ptr
Update pList->pTail to pBlock->pPrevious
Update pList->pHead to pBlock->pNext
Update pBlock->pNext->pPrevious to pBlock->pPrevious
Update pList->pTail to NULL_ptr
Update pList->pTail->pNext to NULL_ptr
Update pList->pHead->pPrevious to NULL_ptr
Update pBlock->pPrevious->pNext to pBlock->pNext
Decrement pList->size

Helper interfaces

helper_LowerProgramBreak()Flow Chart

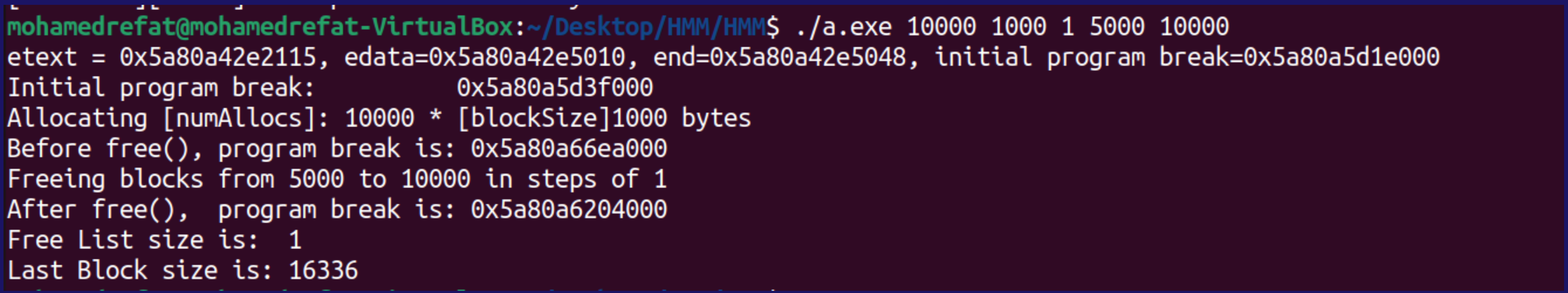


static error_t helper_LowerProgramBreak(freeList_t *pList)
pList is not NULL_ptr?
yes
no
Check if there is no allocated blocks after the last block in the free list
Calculate last address of the last free block
Set kErrorState to kError
pMyHeapTop equals pBlockTopAddress?
yes
no
Get blockLength
There is data allocated after the last free block
Calculate diff between program_break and pMyHeapTop
Calculate total unallocated size unallocated = diff + blockLength
unallocated > SIZE_OF_PAGE*NUM_OF_PAGES
yes
Calculate decrements bytes
sbrk(decrements) != SBRK_ERROR
yes
no
Update last block length
Set kErrorState to kError
Update pMyHeapTop
Update pCurProgBreak

03

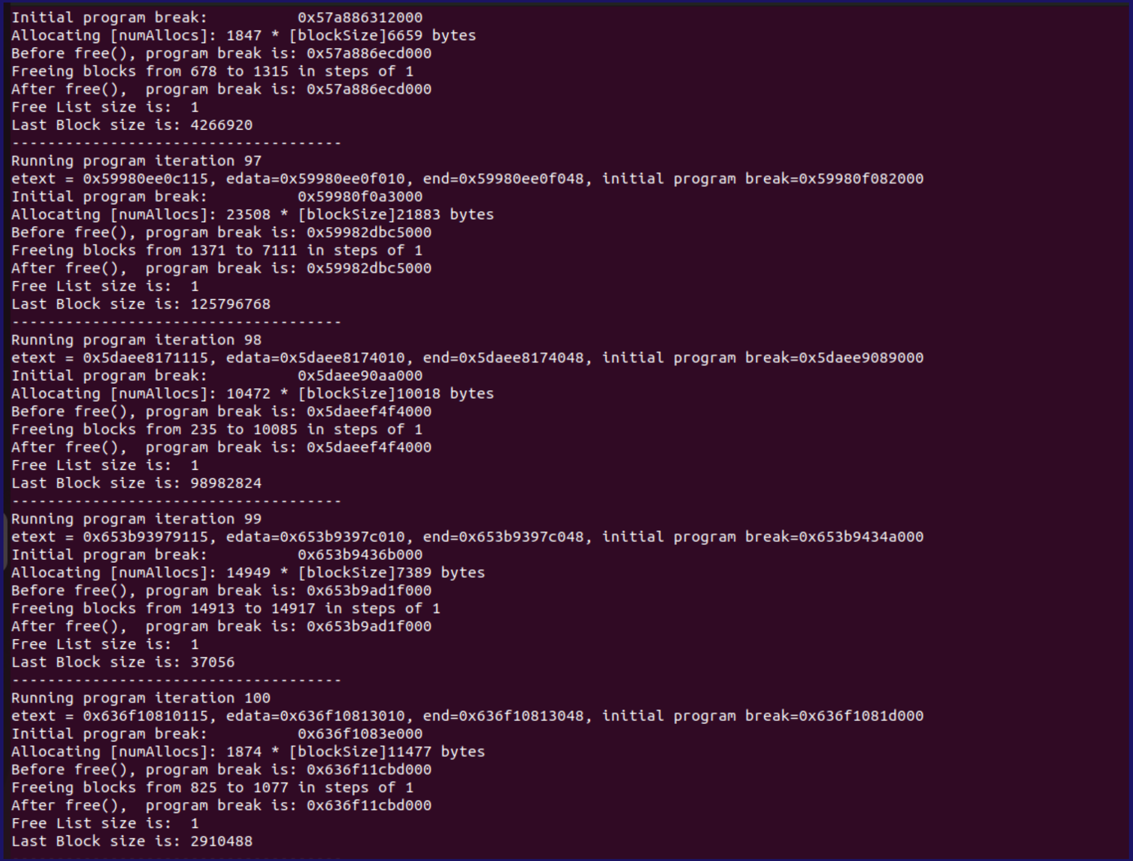
Testing

free_and_sbrk()



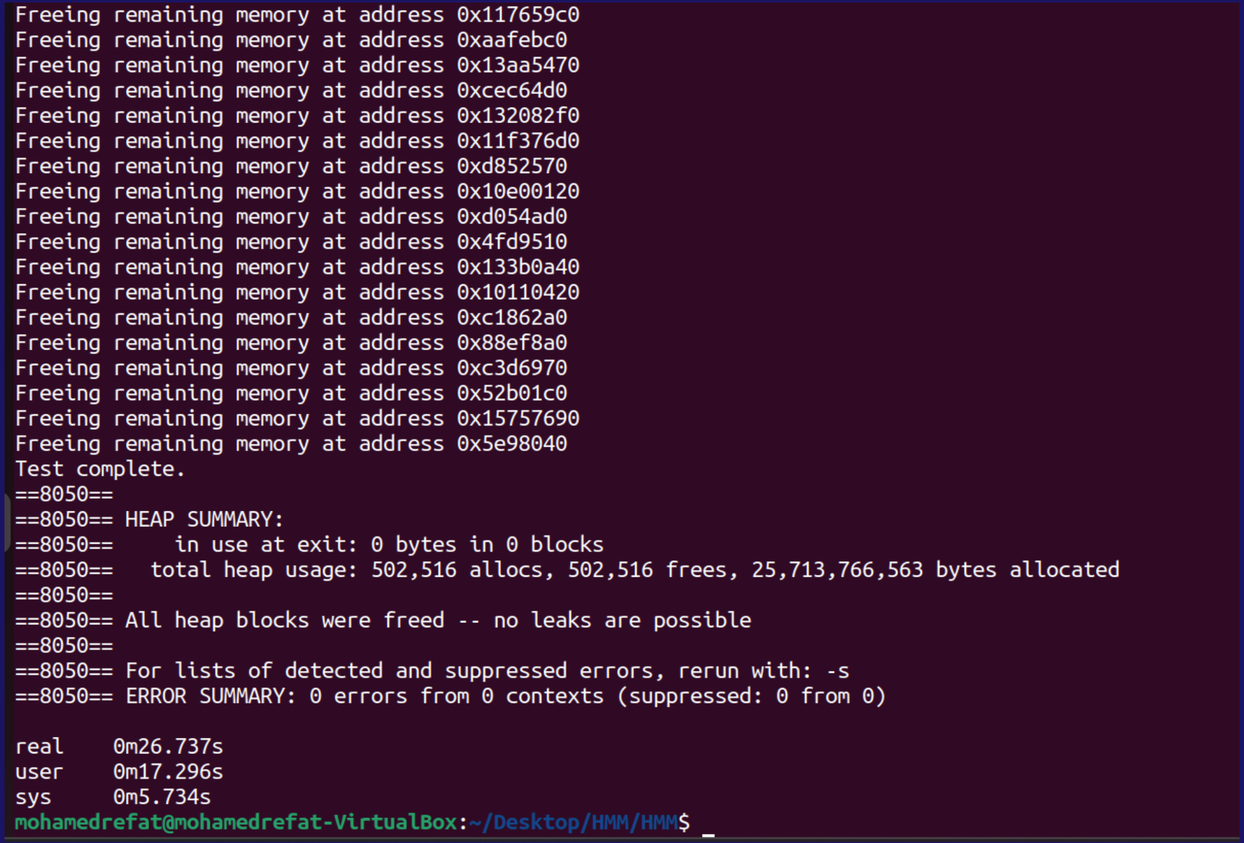
```
mohamedrefat@mohamedrefat-VirtualBox:~/Desktop/HMM/HMM$ ./a.exe 10000 1000 1 5000 10000
etext = 0x5a80a42e2115, edata=0x5a80a42e5010, end=0x5a80a42e5048, initial program break=0x5a80a5d1e000
Initial program break:          0x5a80a5d3f000
Allocating [numAllocs]: 10000 * [blockSize]1000 bytes
Before 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: 0x5daeef4f4000
Freeing blocks from 235 to 10085 in steps of 1
After free(),  program break is: 0x5daeef4f4000
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
```

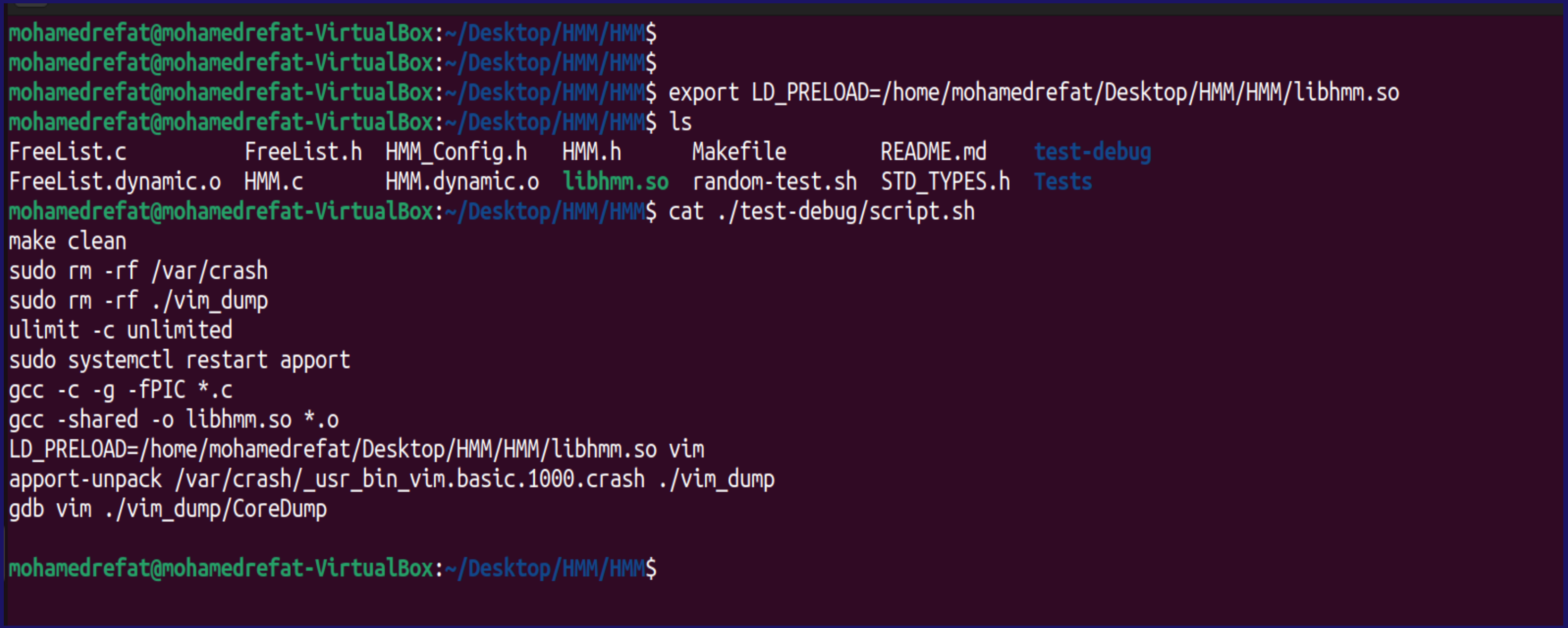
Random allocation and free



```
Freeing remaining memory at address 0x117659c0
Freeing remaining memory at address 0xaafebc0
Freeing remaining memory at address 0x13aa5470
Freeing remaining memory at address 0xcec64d0
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$ _
```

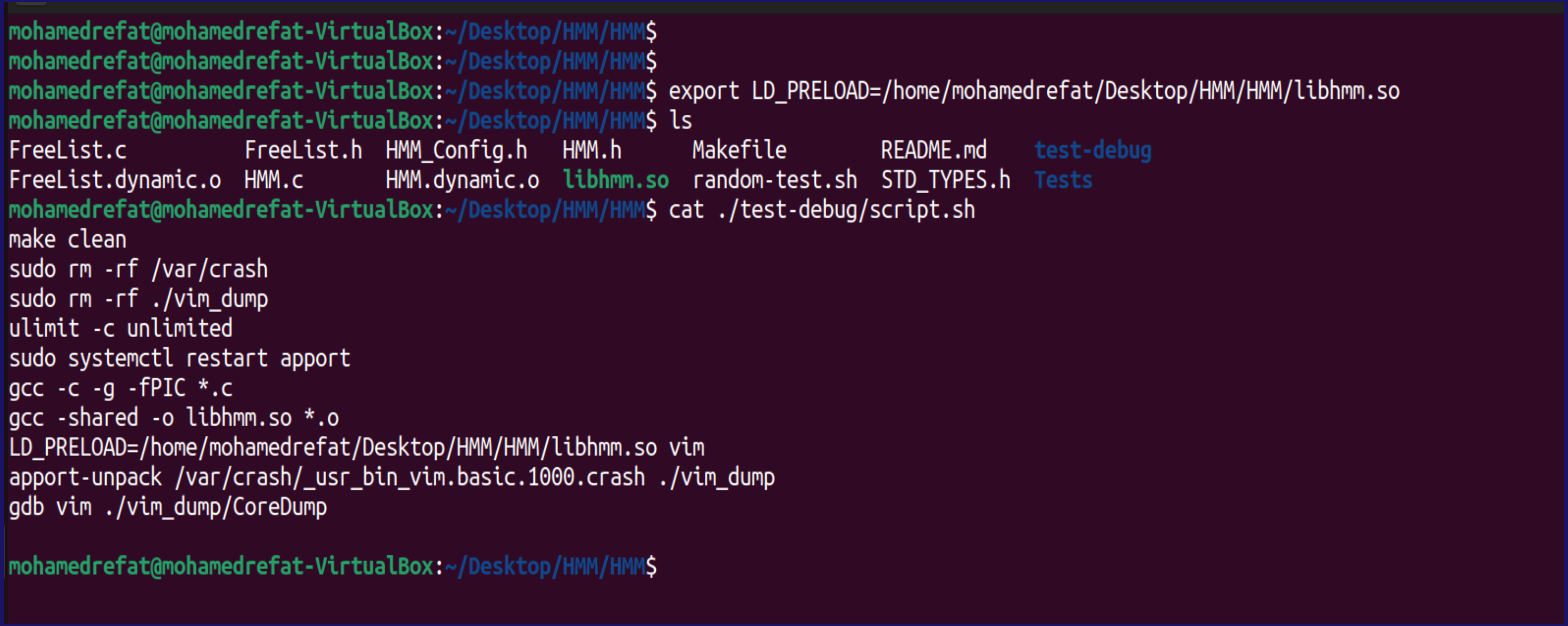
Simple Linux utilities: LS,cat..etc



```
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 woek

THANKS!

Do you have any questions?