Allocator Assignment

Generated by Doxygen 1.8.12

Contents

Chapter 1

Pattern Assignment

1.1 the setup

1.1.1 STEP 1:either use make compile(which run the production versiion) or make debug which runs the debug version.

1.1.2 STEP 2: run ./main to load the program into the main memory.

Chapter 2

Bug List

File log.h

No known Bugs.

File main.c

No Known Bugs

File malloc.c

No Known bugs.

File malloc.h

no knowe bugs

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 Meminfo Struct Reference

this structure contains the declaratiuon of the metadata structure.

```
#include <malloc.h>
```

Public Attributes

- size_t size
- Meminfo ∗ next
- int free

5.1.1 Detailed Description

this structure contains the declaratiuon of the metadata structure.

5.1.2 Member Data Documentation

5.1.2.1 int Meminfo::free

flag to check whether the block is free

5.1.2.2 Meminfo∗ Meminfo::next

holds the address the address of the next metadata

5.1.2.3 size_t Meminfo::size

Size of the block

The documentation for this struct was generated from the following file:

- malloc.h

Chapter 6

File Documentation

6.1 log.h File Reference

This file contains all the debug macros.

Macros

- #define **LOG**(msg)

6.1.1 Detailed Description

This file contains all the debug macros.

This debug macros when enabled in debug mode print the debug messages into the stderr.

Author

Prakash

Bug No known Bugs.

6.2 main.c File Reference

this is the dirver program for the lib functions

```
#include <stdio.h>
#include <stdint.h>
#include "malloc.h"
```

Functions

- int main ()

  this main method for the driver program.

6.2.1 Detailed Description

this is the dirver program for the lib functions

this file provides a testcase to call the function in the library and print heap size after some operation for verification.

Author

Prakash

Bug No Known Bugs

6.2.2 Function Documentation

6.2.2.1 int main ( )

this main method for the driver program.

In this implementation we have assumed that heap size is 16k which is defined in the malloc.h file and each metadata block takes 24 bytes. First it creates 4 integer memory followed by 200 charactes and 4 integers using my_calloc.So the total free memory will be 65536-(16∗2+200+24∗3) = 65232 After that c is realloced to 100 bytes which causes a split in the 200 byte block 100 block which is used and and 100 bloc which is free So the size shhould be 65332 bytes but the actual free space will be 65308 because the 100 byte free block has covered a metadata of 24bytes. Now when free on c is called then it frees the 100 byte block for c and it is merged with the next 100 byte block.So the size now is 65442. Now when free on a is called it free 16 bytes and it menges with the next 200 byte free block getting the total memory size to 65472 bytes.

6.3 malloc.c File Reference

contains all the functions of the allocator.

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "log.h"
#include "malloc.h"
```

Functions

- static Meminfo ∗ get_free_block (size_t s)
  checks whether free block of the mentioned size is available or not.
- static int check_for_enough_space (Meminfo ∗last, size_t s)
  checks whether a block has sufficient memory.
- static void merge_free_block (Meminfo ∗left, Meminfo ∗right)
  memrge to memory block.
- void ∗ my_malloc (size_t size)
  This function allocates memory from the 64KB Array.
- void ∗ my_calloc (size_t num, size_t size)
  this function allocates intialized memory in 64 KB Array.
- void my_free (const void ∗ptr)
  This function free the memory allocated by my_malloc and my_calloc.
- void ∗ my_realloc (void ∗ptr, size_t size)
  This function's reallocates the previous allocated chuck with new size.
- uint32_t free_space_in_my_heap (void)
  this function returns the current free memory of the heap excluding the metadata.

Variables

- static uint8_t heap [MAX_SIZE]
- static Meminfo ∗ head = NULL
- static uint32_t free_space = MAX_SIZE

6.3.1 Detailed Description

contains all the functions of the allocator.

This contains all the implementations of the allocators along with the varouis static utility function whic can only be accessed inside the file.

Author

Prakash

Bug No Known bugs.

6.3.2 Function Documentation

6.3.2.1 static int check_for_enough_space (**Meminfo** ∗ *last*, size_t *s*) [static]

checks whether a block has sufficient memory.

Parameters

last	last segment.
s	size.

Returns

int true or false.

6.3.2.2 uint32_t free_space_in_my_heap (void)

this function returns the current free memory of the heap excluding the metadata.

Parameters

Void	

Returns

uint32_t free size of the heap.

6.3.2.3 static **Meminfo** ∗ get_free_block (size_t *s*) [static]

checks whether free block of the mentioned size is available or not.

Parameters

s	size required.

Returns

Meminfo ∗ pointer to the metadata block.

6.3.2.4 static void merge_free_block (**Meminfo** ∗ *left,* **Meminfo** ∗ *right*) [static]

memrge to memory block.

Parameters

left	left block to be merged.
right	right block.

Returns

void.

6.3.2.5 void∗ my_calloc (size_t *num,* size_t *size*)

this function allocates intialized memory in 64 KB Array.

This function creates the initialized memory using malloc first then using memset to intialize it to zero.

Parameters

num	number of elements.
size	size of each element.

Returns

void ∗ pointer to the data block.

6.3.2.6 void my_free (const void ∗ *ptr*)

This function free the memory allocated by my_malloc and my_calloc.

This function frees up the block allocated by the allocation functions. if the address is not a part of this memory it simply does nothing. finally if there are adjacent block's it merges them to from a new block.

Parameters

ptr	pointer to the memory.

Returns

Void

6.3.2.7 void∗ my_malloc (size_t *size*)

This function allocates memory from the 64KB Array.

This function allocates memory in the 64KB uint_8 array by keeping track of the metadata block followed by the data block for each request and returns NULL if some error happens or it runs out of memory.if there is free block which is large enough then it can split the blockk into two depending on the size of the block that will remain after allocation.

Parameters

size	size to be allocated.

Returns

void ∗ pointer to the data block.

6.3.2.8 void∗ my_realloc (void ∗ *ptr,* size_t *size*)

This function's reallocates the previous allocated chuck with new size.

This ponter to invalid memory is given it return null because to give the user a a error message that it canot be created. if size 0 is passed free is called.if a size less than the cureent size is called then it is allocated and if possible the block is splitted.

Parameters

ptr	pointer to the data
size	new size.

Returns

Void ∗ pointer to the increased size block or null.

6.3.3 Variable Documentation

6.3.3.1 uint32_t free_space = MAX_SIZE `[static]`

free size of the heap

6.3.3.2 Meminfo∗ head = NULL [static]

intial head pointer for the linked list

6.3.3.3 uint8_t heap[MAX_SIZE] [static]

this is the max sizeof the heap∗ 64KB by default

6.4 malloc.h File Reference

This file contains all the memory operations prototype of my library.

Classes

- struct Meminfo

  this structure contains the declaratiuon of the metadata structure.

Macros

- #define MAX_SIZE 65536
- #define FALSE 0
- #define TRUE 1
- #define ALIGN(size) ((((size-1)/4)∗4)+4)

Typedefs

- typedef struct Meminfo **Meminfo**

Functions

- void ∗ my_malloc (size_t size)

  This function allocates memory from the 64KB Array.
- void ∗ my_calloc (size_t num, size_t size)

  this function allocates intialized memory in 64 KB Array.
- void my_free (const void ∗ptr)

  This function free the memory allocated by my_malloc and my_calloc.
- void ∗ my_realloc (void ∗ptr, size_t size)

  This function's reallocates the previous allocated chuck with new size.
- uint32_t free_space_in_my_heap (void)

  this function returns the current free memory of the heap excluding the metadata.

6.4.1 Detailed Description

This file contains all the memory operations prototype of my library.

This contains all the prototypes of the memory operation with metadata structure.

Author

Prakash

Bug no knowe bugs

6.4.2 Macro Definition Documentation

6.4.2.1 #define ALIGN(*size*) ((((size-1)/4)∗4)+4)

this is fo aliginig the space for 4 bytes

6.4.2.2 #define FALSE 0

this is set when block is not free

6.4.2.3 #define MAX_SIZE 65536

this is the max sizeof the heap

6.4.2.4 #define TRUE 1

this is set when block is free

6.4.3 Function Documentation

6.4.3.1 uint32_t free_space_in_my_heap (void)

this function returns the current free memory of the heap excluding the metadata.

Parameters

Void	

Returns

uint32_t free size of the heap.

6.4.3.2 void∗ my_calloc (size_t *num*, size_t *size*)

this function allocates intialized memory in 64 KB Array.

This function creates the initialized memory using malloc first then using memset to intialize it to zero.

Parameters

num	number of elements.
size	size of each element.

Returns

void ∗ pointer to the data block.

6.4.3.3 void my_free (const void ∗ *ptr*)

This function free the memory allocated by my_malloc and my_calloc.

This function frees up the block allocated by the allocation functions. if the address is not a part of this memory it simply does nothing. finally if there are adjacent block's it merges them to from a new block.

Parameters

ptr	pointer to the memory.

Returns

Void

6.4.3.4 void∗ my_malloc (size_t *size*)

This function allocates memory from the 64KB Array.

This function allocates memory in the 64KB uint_8 array by keeping track of the metadata block followed by the data block for each request and returns NULL if some error happens or it runs out of memory.if there is free block which is large enough then it can split the blockk into two depending on the size of the block that will remain after allocation.

Parameters

size	size to be allocated.

Returns

void ∗ pointer to the data block.

6.4.3.5 void∗ my_realloc (void ∗ *ptr,* size_t *size*)

This function's reallocates the previous allocated chuck with new size.

This ponter to invalid memory is given it return null because to give the user a a error message that it canot be created. if size 0 is passed free is called.if a size less than the cureent size is called then it is allocated and if possible the block is splitted.

Parameters

ptr	pointer to the data
size	new size.

Returns

Void ∗ pointer to the increased size block or null.

Index