CS232 Operating Systems
Assignment 03

Faraz Ahmed Khan (fk0398)

Fall 2019

1 Question No. 1

The the implementations of different required functions.

```
Here is the code's␣format␣i␣want␣to␣show␣in␣latex
```

Code from an external file.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  //for close
#include <fcntl.h> // for open
#include <string.h>
#include <sys/mman.h>
#include <math.h>
#include <inttypes.h>
#include "Lec1_st03983_A3_malloc.h"


int TOTAL_SIZE = 1024 * 1024;
int MAGIC = 1234567;
node_t * head = NULL;
void  * start = NULL;


int my_init()
{

    head = mmap(NULL, TOTAL_SIZE, PROT_READ|PROT_WRITE,MAP_ANON|MAP_PRIVATE, -1,

    if (head == MAP_FAILED)
    {
        head = NULL;
        return 0;
    }
    start = head;
    head-> size  = TOTAL_SIZE - sizeof(node_t); //reducing size of node
    head-> next = NULL;
    return 1;
}

void * allocate_on_block(node_t** current, int size)
{
    node_t * temp = *current;
    char * to_move = (char *)*current;
    node * ptr_return = (node *)*current;
    to_move = to_move + size;
    *current = (node_t*) to_move; //moving the given pointer
    (*current)->next = temp->next;
    (*current)->size = temp->size - size;
    ptr_return->size = size - sizeof(node);
    ptr_return->magic = MAGIC;
    return ptr_return + 1;
}

void * my_malloc(int size)
{
    if (head == NULL)
    {
        return NULL;
    }
    size = size + sizeof(node);
    node * ptr = NULL;

    if (head -> size >= size)
    {

       return allocate_on_block(&head, size);
    }
    else
    {
        node_t * temp = head;
        while (temp -> next != 0)
        {
            if (temp -> next -> size >= size)
            {
               return allocate_on_block(&temp->next, size);
            }
            temp = temp->next;
        }

    }

    my_coalesce(); //didnot find memory to allocate hence coalescing

    if (head -> size >= size)
    {

       return allocate_on_block(&head, size);
    }
    else
    {
        node_t * temp = head;
        while (temp -> next != 0)
        {
            if (temp -> next -> size >= size)
            {
               return allocate_on_block(&temp->next, size);
            }
            temp = temp->next;
        }

    }

    printf("failure␣in␣memory␣allocation");
    return 0; //faiture

}

void my_free(void * ptr)
{
    /*
    basically moving the of free list to where we are deleting the node.
    */
    if (ptr == NULL)
    {
        return;
    }
    node* temp = (node *)ptr -1;
    node_t * temp2 = (node_t *) (temp);
    if (temp->magic == MAGIC)
    {

        temp2->size = temp2->size + sizeof(node) - sizeof(node_t);
        temp2->next = head; //putting the head at next.
        head = temp2;
    }
    else{
        printf("Could␣not␣deallocate␣memory");
    }

}

void * my_calloc(int num, int size)
{
    int total_size = num * size;
    char * my_memory = my_malloc(total_size);
    if (my_memory == NULL) //failure in malloc
    {
        printf("Could␣not␣allocate␣memory");
        return NULL;
    }
    for (int i = 0; i < total_size; i++)
    {
        my_memory[i] = 0;
    }
    return my_memory;
}


int min(int num, int num2)
{
    if (num < num2)
    {
        return num;
    }
    return num2;
}
int copy(char * src_ptr, char * dest_ptr, int num_bytes)
{
    /*
    copies num_bytes from scr_ptr to dest_ptr
    */
    int i = 0;
    for (i = 0; i < num_bytes; i++)
    {
        dest_ptr[i] = src_ptr[i];
    }
    return i;
}


void my_coalesce()
{
    /*
    changes the entire structing of the freelist.
    We find the first node, linearly in the memory and use it as head.
    We then move from nodes linearly, i.e. in the order of memory (and not the fr
    and make them adjacent to each in linklist if they can not be coalese, otherw
    */
    node * temp = start;
    head = NULL;

    while (temp->magic == MAGIC)
    {
        temp = (node *)((char *) temp +  temp->size + sizeof(node));
    }
    head = (node_t*) temp;
    node * next_temp = (node *)((char *) temp + temp->size + sizeof(node_t));
    node_t* prev = (node_t*)temp;

    while ((char *)next_temp < (char * ) start + TOTAL_SIZE)
    {
        if (temp->magic != MAGIC && next_temp->magic != MAGIC) //coalescing condi
        {
            temp->size += next_temp->size + sizeof(node_t);
        }
        if (temp -> magic != MAGIC)
        {
            prev = (node_t *)temp;
            prev->next = ((node_t*)(temp))->next;
            temp = next_temp;
        }

        if (next_temp->magic == MAGIC)
        {
            next_temp = (node *)((char * )(next_temp) + next_temp->size + sizeof(
        }
        else{
            next_temp = (node *)((char * )(next_temp) + next_temp->size + sizeof(
        }
    }
    prev->next = 0;

}

void * my_realloc(void * ptr, int size)
{

    if (ptr == NULL)
    {
        return my_malloc(size);
    }
    else if (ptr != NULL && size == 0)
    {
        my_free(ptr);
    }
    else
    {
        node* temp = ptr;
        temp--;
        if (temp->magic == MAGIC)
        {
            int cur_size = temp->size;
            int new_size = size;
            void * new_ptr = my_malloc(new_size);
            int min_size = min(new_size, cur_size);
            int copied_size = copy(ptr, new_ptr, min_size);
            if (copied_size != min_size)
            {
                return NULL;
            }
            my_free(ptr);
            return new_ptr;
        }
    }

    return NULL;
}

void my_showfreelist()
{
    int i = 0;
    node_t * temp = head;
    printf("\n");
    while (temp != 0)
    {
        printf("␣%d:␣%d:␣%" PRIuPTR ":␣%" PRIuPTR "\n", i++, temp->size, (uintptr
        temp = temp->next;
    }
}

void my_uninit()
```

```
{
    if (head != NULL)
    {
        int ret = munmap(start, TOTAL_SIZE);
        if (ret == 0)
        {
            head = start = NULL;
        }
    }
}
```

2 References

Assignment (or part of its logic) was discussed with Mudasir Hanif Shaikh, Huda Feroz Ahmed and Hareem Feroz Ahmed.

Stackoverflow was used for general debugging, and linux manpage was used for official documentation.

3 Comments

The assignment helped in building the concepts regarding memory management, and made me realize the lack of proper internet sources for these topics.