CS232 Operating Systems
Assignment 03: Implementing memory management routines

Syed Muhammad Raza Naqvi (sn03805)

Fall 2019

MakeFile

```
all:
        gcc Lec1_st03805_A3_main.c Lec1_st03805_A3_malloc.c -o main.out

clean:
        rm -rf *.out
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>

#include "Lec1_st03805_A3_malloc.h"

int main()
{
    return 0;
};
```

malloc.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include "Lec1_st03805_A3_malloc.h"


node_t *head = NULL; //points to the start of free list.
node_t *start_mmap = NULL;

int my_init()
{
    if (head != NULL)
    {
        printf("\nRequest␣rejected!␣my_init␣already␣initialized!\n");
```

```
        return 0;
    };
    head = mmap(NULL, memory, PROT_READ|PROT_WRITE, MAP_ANON|MAP_PRIVATE, -1, 0);
    start_mmap = head;
    if (head == MAP_FAILED)
    {
        printf("\nRequest␣rejected!,␣mmap␣call␣unsuccessful!\n");
        return 0;
    };

    head->size = memory - sizeof(node_t);
    head->next = NULL;
    return 1;

};



void my_free(void *ptr)
{
    if (head != NULL)

    {
        if (*((int *)ptr - 1) != magic)
        {
            printf("\nInvalid␣pointer␣given!\n");
        }
        else
        {
            //my_coalesce();
            node_t *temp = head;
            while (temp->next != NULL)
            {
                temp = temp->next;

            };

            int ptr_size = *((int *)ptr - 2);
            int *tptr = (int *)ptr - 2; //ptr casted!
            //printf("\n%p\n",tptr);
            temp->next = (node_t *) tptr;
            temp->next->size = ptr_size;
            temp->next->next = NULL;
        };
    }
    else
    {
        printf("\nmy_free␣unsuccessful,␣my_init␣not␣initialized!\n");
    };

};

void *my_malloc(int size)
{
    if (head == NULL)
```

```
 {
     printf("\nmy_malloc␣unsuccessful,␣my_init␣isnt␣initialized!\n");
     return NULL;
 };
 if (size < 1)
 {
     printf("\nmy_malloc␣unsuccessful,␣size␣can't␣be␣%d!\n", size);
     return NULL;
 };

if ((sizeof(node_t) - 2*sizeof(int) + 1) > size)
 {
     size = (sizeof(node_t) - 2*sizeof(int) + 1);
 };

 if (head->next == NULL)
 {

     int old_head_size = head->size;
     int actual_free_space = sizeof(node_t) + old_head_size;
     int needed_space = (2*sizeof(int)) + size;

     if (actual_free_space > (needed_space + sizeof(node_t)))
     {
         int* ptr = (int*) head;
         *ptr = size;
         *(ptr+1) = magic; //Magic number.
         printf("\nMemory␣of␣size␣%d␣allocated!\n", size);

         if ((actual_free_space - needed_space) >= sizeof(node_t))
         {
             head = (node_t *)((char *) head + needed_space);
             head->size = actual_free_space - needed_space - sizeof(node_t);
             head->next = NULL;
             return ptr + 2;
         };
     }
     else
     {
         printf("\nMemory␣of␣size␣%d␣unavailable!\n",size);
         return NULL;
     };
 };

 if (head->next != NULL)
 {
     node_t* temp = head;
     node_t* prev = head;

     while(temp != NULL)
     {
         int old_temp_size = temp->size;
         int actual_free_space = sizeof(node_t) + old_temp_size;
         int needed_space = (2*sizeof(int)) + size;
```

```
if (actual_free_space == needed_space)
{
    int* ptr = (int*) temp;
    *ptr = size;
    *(ptr+1) = magic; //Magic number.
    printf("\nMemory␣of␣size␣%d␣allocated!\n", size);

    if (temp == head)
    {
        head = head->next;
    };

    if (temp != head)
    {
        prev->next = temp->next;
    };

    return ptr + 2;
};

if (actual_free_space > needed_space)
{
    //printf("entered");
    node_t *temp_next = temp->next;
    int* ptr = (int*) temp;
    *ptr = size;
    *(ptr+1) = magic; //Magic number.
    printf("\nMemory␣of␣size␣%d␣allocated!\n", size);

    if ((actual_free_space - needed_space) >= sizeof(node_t))
    {
        if (head == temp)

        {
            temp = (node_t *)((char *) temp + needed_space);
            int free_size = actual_free_space - needed_space - sizeof
            temp->size = actual_free_space - needed_space - sizeof(no
            //printf("%d",free_size);
            temp->next = temp_next;
            head = temp;
            return ptr + 2;
        }
        else
        {
            temp = (node_t *)((char *) temp + needed_space);
            int free_size = actual_free_space - needed_space - sizeof
            temp->size = actual_free_space - needed_space - sizeof(no
            //printf("%d",free_size);
            temp->next = temp_next;
            prev->next = temp;
            return ptr + 2;
        }
    };

    if ((actual_free_space - needed_space) < sizeof(node_t))
```

```
                {
                    if (temp != head)
                    {
                        prev->next = temp->next;
                        return ptr + 2;
                    }
                    else
                    {
                        head = head->next;
                        return ptr + 2;
                    }
                };
            };

            prev = temp;
            temp = temp->next;
        };
    };
    printf("\nMemory␣of␣size␣%d␣unavailable!\n",size);
    return NULL;
};

void *my_calloc(int num, int size)
{

    if (head == NULL)
    {
        printf("\nmy_calloc␣unsuccessful,␣my_init␣isnt␣initialized!\n");
        return NULL;

    };

    int total_size = num*size;
    //printf("%d",total_size);
    void *p = my_malloc(total_size);
    //printf("\n%d\n",*((int *)p - 1));
    if (p == NULL)
    {
        printf("\ncalloc␣isnt␣successful!\n");

    }
    else
    {
        char *ptr = (char *) p;
        int counter = 0;

        while (counter != total_size)
        {
            *(ptr + counter) = '\0';
            counter++;
        };

    };

    return p;
```

```
};

void *my_realloc(void *old_ptr, int new_size)
{

    if (head == NULL)
    {
        printf("\nmy_realloc unsuccessful, my_init isnt initialized!\n");
        return NULL;

    };

    int old_size = *(((int *) old_ptr) - 2);
    //printf("%d", old_size);
    //int needed_size = old_size + new_size;

    void *new_ptr = my_malloc(new_size);

    if (new_ptr == NULL)
    {
        printf("\nRequest of realloc rejected.\n");
        return NULL;
    };

    int counter = 0;
    int iteration = old_size;

    if (old_size > new_size)
    {
        iteration = new_size;
    };

    while (counter != iteration)
    {
        *((char *) new_ptr + counter) = *((char *) old_ptr + counter);
        counter++;

    };

    printf("\nRequest of realloc accepted!\n");
    my_free(old_ptr);
    return new_ptr;
};




void my_showfreelist()
{
    if (head != NULL)
    {
        int node_no = 0;
        node_t *temp = head;
        printf("\nStart of Freelist\n");
```

```
        while (temp != NULL)
        {
            printf("%d:%d:%p\n", node_no,temp->size,temp);
            node_no++;
            temp = temp->next;
        };
        printf("End␣of␣Freelist\n");
    };

    if (head == NULL)
    {
        printf("\nmy_showfreelist␣unsuccessful,␣my_init␣isnt␣initialized!\n");

    };
};



void my_coalesce()
{
    if (head != NULL)
    {
        node_t *iterator_start;
        node_t *iterator_end;
        node_t *iterator_prev;
        node_t *current_start = head;
        node_t *current_end;
        while (current_start->next!= NULL)
        {
            iterator_start = current_start->next;
            iterator_end = (node_t *)((char *) iterator_start + iterator_start->s

            iterator_prev = current_start;
            current_end = (node_t *)((char *)current_start + current_start->size
            while (iterator_start->next != NULL)
            {

                if (current_start == iterator_end)
                {
                     iterator_start->size = iterator_start->size + current_start-
                     if (iterator_prev == head)
                     {
                        head = head->next;
                        current_start = current_start->next;
                     }
                     else
                     {
                        iterator_prev->next = current_start->next;
                     };

                };

                if (current_end == iterator_start)
                {
                     current_start->size = iterator_start->size + current_start->
```

```
                    if (iterator_prev == head)
                    {
                       head = current_start->next;
                    }
                    else
                    {
                       current_start->next = iterator_start->next;
                    };

                };


                if (iterator_start->next != NULL)
                {
                iterator_start = iterator_start->next;
                iterator_end = (node_t *)((char *) iterator_start + iterator_star
                };

            };

            iterator_prev = current_start;
            current_start = current_start->next;
            //my_showfreelist();

        };
    }
    else
    {
        printf("\nmy_coalesce unsuccessful, my_init isnt initialized!\n");
    };


};



void my_uninit()
{

    if (head == NULL)
    {
        printf("\nmy_uninit unsuccessful, first call my_init!\n");
    }
    else
    {
        int unmap = munmap(start_mmap, memory);
        //printf("\n%d\n",unmap);
        if (unmap == 0)
        {
            printf("\nmy_uninit successful!\n");
            head = NULL;
        }
        else
        {
            printf("\nmy_uninit unsuccessful!\n");
```

```
        };

    };

};
```

malloc.h

```
#ifndef RANDOM
#define RANDOM



#define magic 12345
#define memory 1024
//#define memory 1048576



typedef struct node_t
{
    int size;
    struct node_t *next;

} node_t;

extern node_t *head;
extern node_t *start_mmap;

int my_init();
void my_free(void *ptr);
void *my_malloc(int size);
void *my_calloc(int num, int size);
void *my_realloc(void *old_ptr, int new_size);
void my_showfreelist();
void my_coalesce();
void my_uninit();


#endif
```