CS232 Operating Systems
Assignment 03

Mudasir Hanif Shaikh (ms03831)

Fall 2019

1 malloc.h

```c
/*
Mudasir Hanif Shaikh (ms03831)
CS 2021, Habib University
Assignment 3, OS, Fall 2019
*/

#ifndef _Lec2_ms03831_A3_malloc_h
#define _Lec2_ms03831_A3_malloc_h

typedef struct node {
    int size;
    struct node *next;
} my_node;

int my_init();

void *my_malloc();

void my_free(void*);

void* my_calloc();

void* my_realloc(void *, int);

void my_coalesce();

void my_showfreelist();

void my_uninit();

#endif
```

2 malloc.c

```c
/*
```

```c
Mudasir Hanif Shaikh (ms03831)
CS 2021, Habib University
Assignment 3, OS, Fall 2019
*/

#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/mman.h>
#include "Lec2_ms03831_A3_malloc.h"

#define MEGABYTE 1024*1024

my_node *head = NULL;
char* start = NULL;
const int MAGIC = 1234567;
```

2.1 my_init

```c
int my_init(){
  if (head == NULL){
    void *ret = mmap(NULL, MEGABYTE, PROT_READ|PROT_WRITE, MAP_ANON
    |MAP_PRIVATE, -1, 0);
    if (ret == MAP_FAILED) return 0;
    else{
      head = (my_node *) ret;
      start = (char *) ret;
      head->size = MEGABYTE - sizeof(my_node);
      head->next = 0;
      return 1;
    }
  }
  else return 1;
}
```

2.2 my_malloc

```c
void *my_malloc(int size){
  my_node* current = head;
  my_node* temp = head;
  while (current->size < size + sizeof(my_node) && current->next !=
     NULL){
    temp = current;
    current = current->next;
  }
  if (current->size < size + sizeof(my_node)){
    printf("%s\n", "MALLOC FAILED: not enough memory.");
    errno = ENOMEM;
    return NULL;
  }
  else{
    if (current == head){
      head = (my_node*) (((char*)head) + size + sizeof(my_node));
      head->size = temp->size - size - sizeof(my_node);
      head->next = temp->next;
    }
    else{
      temp->next = (my_node*) ((char *) current + size + sizeof(
    my_node));
      temp->next->next = current->next;
      temp->next->size = current->size - size - sizeof(my_node);
    }
    current->size = size;
    current->next = (my_node*) &MAGIC;
    current++;
    return current;
  }
}
```

2.3 my_free

```
void my_free(void* ptr){
  my_node* freePtr = ((my_node *) ptr) - 1;
  if (freePtr->next == (my_node*) &MAGIC){
     my_node* previousHead = head;
       head = freePtr;
       head->size = freePtr->size;
       head->next = previousHead;
  }
  else{
    printf("%s\n", "The pointer passed to free is not valid");
    return;
  }
}
```

2.4 my_calloc

```
void* my_calloc(int num, int size){
  my_node* t = my_malloc(num*size);
  if (t != NULL){
    char* temp = (char*) t;
    for (int i = 0; i < num*size; i++) {
      *temp = 0;
      temp++;
    }
    return t;
  }
  else{
    printf("%s\n", "my_calloc failed: not enough memory.");
    return NULL;
  }
}
```

2.5 my_realloc

```c

void* my_realloc(void * ptr, int size){
  if (size < 0){
    printf("%s\n", "Please specify a valid size to reallocate");
    return NULL;
  }

  if (ptr == NULL){
    return my_malloc(size);
  }

  if (size == 0){
    my_free(ptr);
    return NULL;
  }
  else if (size > 0){
    my_node* temp = (my_node*) ptr;
    int previousSize = (temp - 1)->size;
    if ((temp - 1)->next == (my_node*)&MAGIC) {
      my_node* newPtr = my_malloc(size);
      if (newPtr){

        char* previousMem = (char*) temp;
        char* newMem = (char*) newPtr;
        int minimumSize = previousSize;
        if (size < minimumSize){
          minimumSize = size;
        }
        for (int i = 0; i < minimumSize; i++){
          *newMem = *previousMem;
          newMem++;
          previousMem++;
        }
        my_free(ptr);
        return newPtr;
      }
      else{
        printf("reallocate failed \n");
        return NULL;
      }
    }
    else{
      printf("%s\n", "Pointer is not valid");
      return NULL;
    }
  }
  else{
    return NULL;
  }
}
```

2.6 my_coalesce

```c

void my_coalesce(){
  my_node* starting_node = (my_node*)start;
  my_node* next_node_in_heap = (my_node*) (((char*) (starting_node + 1)) + starting_node->size);
  head = NULL;
  my_node* prev = NULL;
  while (next_node_in_heap < ((my_node*)(start + MEGABYTE))) {
    if (starting_node->next == (my_node*)&MAGIC){
      starting_node = (my_node*) (((char*) (starting_node + 1)) + starting_node->size);
      continue;
    }
    else{
      next_node_in_heap = (my_node*) (((char*) (starting_node + 1)) + starting_node->size);
      if (next_node_in_heap + 1 > ((my_node*)(start + MEGABYTE))) {
        return;
      }
      if (head == NULL){
        head = starting_node;
        head->next = NULL;
        head->size = starting_node->size;
        prev = head;
        while (next_node_in_heap->next != (my_node*)&MAGIC) {
          head->size = head->size + next_node_in_heap->size + sizeof(my_node);
          next_node_in_heap = (my_node*) (((char*) (next_node_in_heap + 1)) + next_node_in_heap->size);
          starting_node = (my_node*) (((char*) (starting_node + 1)) + starting_node->size);
          if (starting_node + 1 > ((my_node*)(start + MEGABYTE))) {
            return;
          }
        }
      }
      else{
        prev->next = starting_node;
        while (next_node_in_heap->next != (my_node*)&MAGIC) {
          prev->next->size =  prev->next->size + next_node_in_heap->size + sizeof(my_node);
          next_node_in_heap = (my_node*) (((char*) (next_node_in_heap + 1)) + next_node_in_heap->size);
          starting_node = (my_node*) (((char*) (starting_node + 1)) + starting_node->size);
          if (starting_node + 1 > ((my_node*)(start + MEGABYTE))){
            return;
          }
        }
      }
    }
  }
  return;
}
```

2.7 my_showfreelist

```c
void my_showfreelist(){
  my_node* current = head;
  int no = 1;
  while (current != NULL){
    printf("%d: %d: %p\n", no, current->size, (void *) current);
    no++;
    current = current->next;
  }
}
```

2.8 my_uninit

```
void my_uninit(){
  if (start != NULL){
    munmap(start, MEGABYTE);
    return;
  }
}
```

References

[1] Collaborated with Kainat Abbasi and Rayyan ul Haq on parts of this assignment.

[2] The Linux Programming Interface

[3] Free Space Management, Operating Systems - Three Easy Pieces