ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Сибирский государственный университет
телекоммуникаций и информатики»
(СибГУТИ)

Отчет по лабораторной работе №2
по дисциплине “Системное программное обеспечение”

Выполнил:
студент гр. ИС-441
Брагин А.С.
Проверил:
Мамойленко С. Н.

Подпись_____

Дата_____

Новосибирск, 2017

Задание на лабораторную работу

1. Напишите программу, которая выделяет 1000 блоков памяти по 120 байт каждый и выводит статистику (malloc_stats). Сколько блоков памяти было выделено в разделе «малых блоков»? Измените программу так, чтобы размер блока рассчитывался как i*1024, где i – номер итерации цикла. Повторите эксперимент и ответьте на тот же вопрос.

2. Напишите программу, которая перехватывая функций malloc и free реализует собственную систему управления динамической памяти (кучу)[5]. Память для кучи может выделяться как стандартной функцией malloc, так и с помощью вызова mmap. Программа поддерживает выделение блоков размером в диапазоне от «ГР» до «МР*ГР», где МР – номер месяца Вашего рождения, ГР – год Вашего рождения. Выделение блоков других размеров недопустимо. Максимальный размер кучи задается параметром командной строки. Блоки выделяются по принципу «первый подходящий». Свободные блоки хранятся в виде одного двунаправленного несортированного списка.

Результаты

Программа, которая выделяет 1000 блоков памяти по 120 байт каждый и выводит статистику (malloc_stats).

```c
#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>

#define INTERNAL_SIZE_T size_t
#define SIZE_SZ (sizeof(INTERNAL_SIZE_T))
#define MALLOC_ALIGN_MASK 2 * SIZE_SZ
#define MIN_CHUNK_SIZE (offsetof(struct malloc_chunk, fd_nextsize))
#define MINSIZE (unsigned long)(((MIN_CHUNK_SIZE+MALLOC_ALIGN_MASK)&~MALLOC_ALIGN_MASK))
#define chunk2mem(p) ((void*)((char*)(p) + 2*SIZE_SZ))
#define mem2chunk(mem) ((mchunkptr)((char*)(mem) - 2*SIZE_SZ))

struct malloc_chunk
{
    INTERNAL_SIZE_T prev_size;
    INTERNAL_SIZE_T size;
    struct malloc_chunk* fd;
    struct malloc_chunk* bk;
    struct malloc_chunk* fd_nextsize;
    struct malloc_chunk* bk_nextsize;
};

typedef struct malloc_chunk* mchunkptr;

#define request2size(req) (((req) + SIZE_SZ + MALLOC_ALIGN_MASK < MINSIZE) ? MINSIZE : ((req) + SIZE_SZ + MALLOC_ALIGN_MASK) & ~MALLO

int main()
{

    int count = 1000;
    int size = 120;
    int *p[count];

    for (int i = 0; i < count; i++) {
        p[i] = malloc(request2size(size));
        printf ("Block's size = %ld\n", (mem2chunk(p[i])->size)&(~0x7));
    }

    malloc_stats();

```

Результаты:

```
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
Block's size = 144
```

Программа изменённая так, чтобы размер блока рассчитывался как i*1024, где i – номер итерации цикла.

```c
#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>

#define INTERNAL_SIZE_T size_t
#define SIZE_SZ (sizeof(INTERNAL_SIZE_T))
#define MALLOC_ALIGN_MASK 2 * SIZE_SZ
#define MIN_CHUNK_SIZE (offsetof(struct malloc_chunk, fd_nextsize))
#define MINSIZE (unsigned long)(((MIN_CHUNK_SIZE+MALLOC_ALIGN_MASK)&~MALLOC_ALIGN_MASK))
#define chunk2mem(p) ((void*)((char*)(p) + 2*SIZE_SZ))
#define mem2chunk(mem) ((mchunkptr)((char*)(mem) - 2*SIZE_SZ))

struct malloc_chunk
{
    INTERNAL_SIZE_T prev_size;
    INTERNAL_SIZE_T size;
    struct malloc_chunk* fd;
    struct malloc_chunk* bk;
    struct malloc_chunk* fd_nextsize;
    struct malloc_chunk* bk_nextsize;
};

typedef struct malloc_chunk* mchunkptr;

#define request2size(req) (((req) + SIZE_SZ + MALLOC_ALIGN_MASK < MINSIZE) ? MINSIZE : ((req) + SIZE_SZ + MALLOC_ALIGN_MASK) & ~MALLO

int main()
{
....
    int count = 1000;
    int size = 1024;
    int *p[count];
....
    for (int i = 0; i < count; i++) {
        p[i] = malloc(request2size(i * size));
        printf ("Block's size = %ld\n", (mem2chunk(p[i])->size)&(~0x7));
    }
.................
    malloc_stats();
........
 1Help  2Save  3Mark  4Replac  5Copy  6Move  7Search  8Delete  9PullDn  10Quit
```

Результаты:

```
anton@anton-VirtualBox:~/Documents/SisOS$ ./l2
Block's size = 48
Block's size = 1040
Block's size = 2064
Block's size = 3088
Block's size = 4112
Block's size = 5136
Block's size = 6160
Block's size = 7184
Block's size = 8208
Block's size = 9232
Block's size = 10256
Block's size = 11280
Block's size = 12304
Block's size = 13328
Block's size = 14352
Block's size = 15376
Block's size = 16400
Block's size = 17424
Block's size = 18448
Block's size = 19472
Block's size = 20496
Block's size = 21520
Block's size = 22544
Block's size = 23568
Block's size = 24592
Block's size = 25616
Block's size = 26640
Block's size = 27664
Block's size = 28688
Block's size = 29712
Block's size = 30736
Block's size = 31760
Block's size = 32784
Block's size = 33808
Block's size = 34832
Block's size = 35856
Block's size = 36880
Block's size = 37904
Block's size = 38928
Block's size = 39952
```

```
Block's size = 1015808
Block's size = 1015808
Block's size = 1015808
Block's size = 1015808
Block's size = 1019904
Block's size = 1019904
Block's size = 1019904
Block's size = 1019904
Block's size = 1024000
Block's size = 1024000
Block's size = 1024000
Block's size = 1024000
Arena 0:
system bytes     =    8331264
in use bytes     =    8326192
Total (incl. mmap):
system bytes     =  513728512
in use bytes     =  513723440
max mmap regions =        872
max mmap bytes   =  505397248
```

Программа, которая перехватывает функций malloc и free.

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

#define INTERNAL_SIZE_T size_t
#define SIZE_SZ (sizeof(INTERNAL_SIZE_T))
#define MALLOC_ALIGN_MASK 11*1996
#define MIN_CHUNK_SIZE (offsetof(struct malloc_chunk, fd_nextsize))
#define MINSIZE 1996
#define ALLSIZE 1000*MALLOC_ALIGN_MASK
#define chunk2mem(p) ((void*)((char*)(p) + 2*SIZE_SZ))
#define mem2chunk(mem) ((mchunkptr)((char*)(mem) - 2*SIZE_SZ))

int FIRST_FLAG = 0;

struct malloc_chunk
{
    INTERNAL_SIZE_T prev_size;
    INTERNAL_SIZE_T size;
    struct malloc_chunk* prev;
    struct malloc_chunk* next;
};

typedef struct malloc_chunk* mchunkptr;
mchunkptr HEAD;

#define request2size(req) (((req) + SIZE_SZ + MALLOC_ALIGN_MASK < MINSIZE) ?
MINSIZE : ((req) + SIZE_SZ + MALLOC_ALIGN_MASK) & ~MALLOC_ALIGN_MASK)


static void *(*old_malloc_hook) (size_t, const void *);
static void (*old_free_hook) (void *, const void *);

static void my_init_hook();
static void *my_malloc_hook(size_t size, const void *caller);
static void my_free_hook (void *ptr, const void *caller);

void (* volatile __malloc_initialize_hook) (void) = my_init_hook;


int main()
{
    void* p[3];

    p[0] = malloc(10);
    p[1] = malloc(2000);
    p[2] = malloc(12*1996);

    printf("\nHEAD  %p  p[0]  %p  p[1]  %p  p[2]  %p\n\n", HEAD, p[0], p[1],
p[2]);

    malloc_stats();
    printf("\n\n");

    for(int i = 0; i < 3; i++) free(p[i]);

    return 0;
}

static void my_init_hook()
{
```

```
    old_malloc_hook = __malloc_hook;
    old_free_hook = __free_hook;
    __malloc_hook = my_malloc_hook;
    __free_hook = my_free_hook;
}

static void *my_malloc_hook(size_t size, const void *caller)
{
    void *result;
    __malloc_hook = old_malloc_hook;

    if(FIRST_FLAG == 0){
          FIRST_FLAG = 1;
          HEAD = malloc(ALLSIZE);
          HEAD->prev_size = 0;
          HEAD->size = 1;
          HEAD->next = HEAD + 1;
          HEAD->next->prev_size = 1;
          HEAD->next->size = ALLSIZE - 1;
          HEAD->next->prev = HEAD;
          HEAD->next->next = HEAD + ALLSIZE;
    }

    size = request2size(size);

    struct malloc_chunk* node = HEAD->next;

    while (node < HEAD + ALLSIZE){
          if(node->size >= size){
                struct malloc_chunk* node_t = node + size;
                node_t->size = node->size - size;
                if ((node_t > HEAD + ALLSIZE) || (node_t->size < MINSIZE)) {
                      node->size = size;
                      node->prev = 0;
                      node->prev = 0;
                      node->next = 0;
                      result = node;

                      old_malloc_hook = __malloc_hook;
                      __malloc_hook = my_malloc_hook;
                      return result;
                }
                else {
                      node->size = size;
                      node->prev->next = node_t;
                      node_t->prev_size = node->size;
                      node_t->prev = node->prev;
                      node_t->next = node->next;
                      node->prev = 0;
                      node->next = 0;
                      result = node;

                      old_malloc_hook = __malloc_hook;
                      __malloc_hook = my_malloc_hook;
                      return result;
                }
          }
          else node = node->next;
    }

    old_malloc_hook = __malloc_hook;
    __malloc_hook = my_malloc_hook;

    return 0;
}
```

```
static void my_free_hook (void *ptr, const void *caller)
{
    __free_hook = old_free_hook;

    struct malloc_chunk* node = ptr;
    struct malloc_chunk* node_t = node + node->size;
    int f = 0;

    //prev | freeing | next free
    if (node_t->next != 0){
          f = 1;

          //plus
          node->size += node_t->size;
          node->next = node_t->next;
          node->prev = node_t->prev;

          //fix
          if(node_t->next < HEAD + ALLSIZE) {
                node_t->next->prev = node;
                struct malloc_chunk* node_z = node_t + node_t->size;
                node_z->prev_size = node->size;
          }

          //free next
          node_t->prev_size = 0;
          node_t->size = 0;
          node_t->prev = 0;
          node_t->next = 0;
    }

    node_t = node - node->prev_size;

    //prev free | freeing | next
    if (node_t->next != 0){
          f = 1;

          //plus
          node_t->size += node->size;

          //free
          node->prev_size = 0;
          node->size = 0;
    }

    //prev use | freeing | next use
    if (f == 0){

          //find free and fix
          struct malloc_chunk* node_z = HEAD;
          while(1) {
                if(node_z->next > node){
                      node->prev = node_z;
                      node->next = node_z->next;
                      node_z->next = node;
                      node_z->next->prev = node;
                      break;
                } else {
                      node_z = node_z->next;
                }
          }
    }

    old_free_hook = __free_hook;
    __free_hook = my_free_hook;
}
```

Результаты

```
HEAD  0x7f4752d8e010  p[0]  0x7f4752d8e030  p[1]  0x7f4752d8ea30  p[2]  0x7f4752d8ec30

Arena 0:
system bytes     =          0
in use bytes     =          0
Total (incl. mmap):
system bytes     =   21958656
in use bytes     =   21958656
max mmap regions =          1
max mmap bytes   =   21958656
```