**重 庆 大 学**

**学 生 实 验 报 告**

**实验课程名称 操作系统**

**开课实验室 DS1501**

**学 院 软件学院 年级**2020 **专业班 软件工程1 班**

**学 生 姓 名 李易燔 学 号 20205644**

**开 课 时 间 2021 至 2022 学年第 二 学期**

|  |  |
| --- | --- |
| **总 成 绩** |  |
| **教师签名** | **刘寄** |

**软件学院制**

**《操作系统原理》实验报告**

**开课实验室： 2022年 4月 23日**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | | 软件学院 | 年级、专业、班 | | 2020级软件工程1班 | 姓名 | | 李易燔 | | 成绩 |  |
| 课程  名称 | | 操作系统 | | 实验项目  名 称 | 内存管理 | | 指导教师 | | 刘寄 | | |
| 教师评语 | 教师签名：刘寄  年 月 日 | | | | | | | | | | |
| **1．实验目的：**   * 掌握内存分配器 * 实现自己的malloc/free   **2．实验内容：**   * + 实现首次/最佳/最坏中的一种分配算法   + 编辑文件userapp/myalloc.c，实现如下四个接口函数     - malloc/free/calloc/realloc   **3．实验报告：**  **myalloc.c文件**  /\*\*  \* vim: filetype=c:fenc=utf-8:ts=4:et:sw=4:sts=4  \*/  #include <sys/types.h>  #include <string.h>  #include <stdint.h>  struct chunk {  char signature[4]; /\* "OSEX" \*/  struct chunk \*next; /\* ptr. to next chunk \*/  int state; /\* 0 - free, 1 - used \*/  #define FREE 0  #define USED 1  int size; /\* size of this chunk \*/  };  static struct chunk \*chunk\_head;  void init\_memory\_pool(size\_t heap\_size, uint8\_t \*heap\_base)  {  chunk\_head = (struct chunk \*)heap\_base;  strncpy(chunk\_head->signature, "OSEX", 4);  chunk\_head->next = NULL;  chunk\_head->state = FREE;  chunk\_head->size = heap\_size;  }  //实验五  void\* g\_heap;  void\* tlsf\_create\_with\_pool(uint8\_t\* heap\_base, size\_t heap\_size)  {  chunk\_head = (struct chunk\*)heap\_base;  strncpy(chunk\_head->signature,"OSEX", 4);  chunk\_head->next = NULL;  chunk\_head->state = FREE;  chunk\_head->size = heap\_size;  return NULL;  }  struct chunk add\_block(int size) {  struct chunk\* a;  struct chunk b;  a = &b;  a->state = FREE;  a->size = size;  a->next = NULL;  strncpy(a->signature, "OSEX", 4);  return b;  }  struct chunk\* findFreeblock(size\_t size) {  struct chunk\* a = chunk\_head;  while (a != NULL) {  if (a->size >= size && a->state == FREE) {  return a;  }  a = a->next;  }  return NULL;  }  void \*malloc(size\_t size)//分配大小为size字节的内存块，并返回块起始地址,如果size是0，返回NULL  {  if (size != 0) {  struct chunk\* tmp = findFreeblock(size);  if (tmp == NULL)return NULL;  else {  void\* ptr;  if (strncmp(tmp->signature, "OSEX", 4) == 0)  ptr = (uint8\_t\*)tmp + sizeof(struct chunk);  else  return NULL;  int freesize= tmp->size - size - sizeof(struct chunk);  if (freesize <= 0) {  tmp->state = USED;  return ptr;  }  else {  struct chunk\* new = (struct chunk\*)((uint8\_t\*)ptr + size);  \*new = add\_block(freesize);  tmp->size = size;  tmp->state = USED;  new->next = tmp->next;  tmp->next = new;  return ptr;//相当于插入链表  }  }  }  else  {  return NULL;  }  }  void free(void \*ptr)//释放ptr指向的内存块,如果ptr是NULL，直接返回  {  if (ptr != NULL) {  struct chunk\* target\_free = (struct chunk\*)(((uint8\_t\*)ptr) - sizeof(struct chunk));  if (strncmp(target\_free->signature, "OSEX", 4) != 0)//验证签名  return;  if (target\_free != NULL) { //进行合并  target\_free->state = FREE;  struct chunk\* curr\_chunk = chunk\_head;  while (curr\_chunk != NULL) {  struct chunk\* curr\_next = curr\_chunk->next;  if (curr\_chunk->state == FREE) {  size\_t size = curr\_chunk->size;  while (curr\_next->state == FREE && curr\_next != NULL) {  size = size + (curr\_next->size + sizeof(struct chunk));  curr\_next = curr\_next->next; //其实只要进入该代码一次即可，可以设置flag判断有无进入。  //也可以先找到target\_free的前一个节点和后一个节点，对他们进行判断合并。  }  curr\_chunk->next = curr\_next;  curr\_chunk->size = size;  }  curr\_chunk = curr\_next;  }  }  }  else  return;  }  void \*calloc(size\_t num, size\_t size)//为num个元素的数组分配内存，每个元素占size字节，把分配的内存初始化成0  {  size\_t sizetotal = num \* size;  void\* ptr = malloc(sizetotal);  if (ptr == NULL) {  return NULL;  }  else {  size\_t i;  for (i = 0; i < sizetotal; i++)  {  \*((uint8\_t\*)ptr + i) = 0;  }  return ptr;  }  }  //重新分配oldptr指向的内存块，新内存块有size字节  //如果oldptr是NULL，该函数等价于malloc(size)  //如果size是0，该函数等价于free(oldptr)  //把旧内存块的内容复制到新内存块  //如果新内存块比较小，只复制旧内存块的前面部分  //如果新内存块比较大，复制整个旧内存块，而且不用初始化多出来的那部分  //如果新内存块还在原来的地址oldptr，返回oldptr；否则返回新地址  //必须验证oldptr的有效性  //必须合并相邻的空闲块  void \*realloc(void \*oldptr, size\_t size)  {  if (size != 0) {  void\* ptr = malloc(size);//先分配一块空间  if (oldptr == NULL)return ptr;  else {  struct chunk\* oldchunk = (struct chunk\*)(((uint8\_t\*)ptr) - sizeof(struct chunk));//找到块头  if (strncmp(oldchunk->signature, "OSEX", 4) != 0)return NULL;  if (oldchunk != NULL) {  int curr\_size;  if (oldchunk->size > size) {  curr\_size = size;  }  else {  curr\_size = oldchunk->size;  }  int i;  for (i = 0; i < curr\_size; i++) {  \*((uint8\_t\*)ptr + i) = \*((uint8\_t\*)oldptr + i);  }  free(oldptr);  return ptr;  }  }  return ptr;  }  else {  free(oldptr);  return NULL;  }  }    进行测试（在github中找到的测试程序）  Step1:在myalloc.c中增加头文件：  #include <unistd.h>  #include <syscall.h>  #include <stdio.h>  随后在myalloc.c中直接编写test\_alloc()函数      Main主函数如下：  /\*  \* vim: filetype=c:fenc=utf-8:ts=4:et:sw=4:sts=4  \*/  #include <inttypes.h>  #include <stddef.h>  #include <math.h>  #include <stdio.h>  #include <sys/mman.h>  #include <syscall.h>  #include <netinet/in.h>  #include <stdlib.h>  #include "graphics.h"  #include<time.h>  extern void\* tlsf\_create\_with\_pool(void\* mem, size\_t bytes);  extern void\* g\_heap;  /\*\*  \* GCC insists on \_\_main  \* http://gcc.gnu.org/onlinedocs/gccint/Collect2.html  \*/  void \_\_main()  {  size\_t heap\_size = 32 \* 1024 \* 1024;  void\* heap\_base = mmap(NULL, heap\_size, PROT\_READ | PROT\_WRITE, MAP\_PRIVATE | MAP\_ANON, -1, 0);  g\_heap = tlsf\_create\_with\_pool(heap\_base, heap\_size);  }  void main(void\* pv)  {    extern void test\_allocator();  test\_allocator();  sleep(1000);  while (1);  task\_exit(0);  }  运行结果： | | | | | | | | | | | |