# 可变分区存储管理

## 1. 算法思想

循环首次适应法

- 把空闲表设计成双向链接结构的循环队列，各空闲区仍按地址从低到高的次序登记在空闲区的管理队列中。
- 同时需要设置一个起始查找指针，指向循环队列中的一个空闲区表项。
- 循环首次适应法分配时总是从起始查找指针所指的表项开始查找。
- 第一次找到满足要求的空闲区时，就分配所需大小的空闲区，使其指向队列中被分配的后面的那块空闲区。
- 下次分配时就从新指向的那块空闲区开始查找。

## 2. 模块设计

### 2.1 主函数 main()

程序的入口，负责初始化数据结构，开启读取和处理输入命令的循环，并调用分配空间函数和释放空间函数，以及接收到退出命令后释放系统函数分配的空间。

### 2.1 初始化函数 initialize()

用 malloc 向系统申请一段空间，并初始化一个头节点 head，作为指向初始时刻空闲区的指针。

### 2.3 分配空间函数 lmalloc()

首先判断所请求空间是否超过最大空间，然后从 head 所指表项开始寻找满足条件的空闲区，更新相应项的地址与大小。若不能找到满足要求的项，则返回空指针。

### 2.4 释放函数 lfree()

首先判断所需释放区段是否超过最大范围，是否与各空闲区存在交叉部分。如果是，则不能

释放空间。否则可以释放相应空间，并根据以下四种情况更新表项：

a) 仅与前空闲区相邻

b) 仅与后空闲区相邻

c) 与前、后空闲区都相邻

d) 与前、后空闲区都不相邻

其中情况 a)、b)只需要将原有节点的信息更新，情况 c)需要将前、后空闲区其中的一个节点删除，合并为一个大的区域，情况 d)需要新建节点并插入链表中。

### 2.5 打印信息函数 log_msg()

检查是否还存在大小为 0 的无效空闲表项，或者两个空闲表项相邻的冗余情况，如果存在就删除相应节点。从 head 所指的节点开始遍历链表，打印每个空闲区的信息。

### 2.6 系统释放函数 free_all()

通过调用库函数 free()释放链表中新建节点时所申请的空间。

## 3. 数据结构与变量

### 3.1 空闲区

```
 5 typedef struct _map
 6 {
 7     unsigned size;
 8     char* addr;
 9     struct _map *next, *prior;
10 } map;
11 //维护一个全局变量头节点head
12 map *head;
```

空闲区表项 map，成员包含了该空闲区的起始地址，大小，以及队列中的上一个和下一个表项的指针。以及全局变量头节点，指向当前空闲表区。

### 3.2 宏定义向系统申请的空间

```
4 #define CHARSIZE 1000
```

### 3.3 基地址

```
13 //基地址
14 char* base;
```

用于保存系统函数 malloc()返回的地址，以此为基准进行分配空间和释放空间的操作，便于阅读和测试。

### 4. 源程序

程序主入口，初始化，读取命令循环，分配和释放空间等操作。

```
228 int main()
229 {
230 //  initialize head
231     initialize();
232 
233     char command;
234     char c;
235     unsigned size;
236     long addr;
237     printf("$command:~");
238     command = getchar();
239     while (command != 'q')
240     {
241         c = getchar();
242         scanf("%u", &size);
243         if (command == 'm')
244         {
245             addr = lmalloc(size);
246             if (addr)
247             {
248                 printf("alloc %u bytes at address %ld.\n", size, (long)addr-(long)base);
249             }
250             else
251             {
252                 printf("memory not enough.\n");
253             }
254             log_msg();
255         }
256         else
257         {
258             c = getchar();
259             scanf("%ld", &addr);
260             lfree(size, addr+(long)base);
261             log_msg();
262         }
263         c = getchar();  //read '\n' from the last command
264         printf("$command:~");
265         command = getchar();
```

分配空间和释放空间函数的头部

```
86 char* lmalloc(unsigned size)
87 {
88     char* ret_addr = NULL;
89 //  size超过最大内存直接返回NULL
90     if (size > CHARSIZE)
91     {
92         return NULL;
93     }
94 //  循环寻找足够的空闲区
```

```
123 void lfree(unsigned size, char* addr)
124 {
125     map* current = head;
126 //  寻找包含addr的空闲区
127     //地址越界
128     if (addr < base || addr + size > base + CHARSIZE)
129     {
130         printf("address out of bound.\n");
131         return;
132     }
133
```

初始化和释放系统空间

```
16 void initialize()
17 {
18 //  initialize map
19     head = malloc(sizeof(map));
20 //  initialize char
21     head->addr = (char*)malloc(CHARSIZE * sizeof(char));
22     head->size = CHARSIZE;
23     head->next = head;
24     head->prior = head;
25     base = head->addr;与前、后空闲区都相邻
26 }
27
28 //释放系统函数malloc申请的内存
29 void free_all()
30 {
31     while (head)
32     {
33         head = head->next;
34         free(head->prior);
35     }
36     return;
37 }
```

## 5. 测试与分析

gcc 编译生成可执行文件 main 后进行调试

```
wzf@wzf:~/courses/OS$ cat run
rm main
gcc -std=c99 -g -o main main.c
gdb ./main
```

### 测试 1

测试样例

```
wzf@wzf:~/courses/OS$ cat test
m 900
f 100 0
m 20
f 100 700
f 100 300
f 100 400
f 100 600
f 100 500
q
```

输出

```
wzf@wzf:~/courses/OS$ ./main < test.txt
$command:~alloc 900 bytes at address 0.
available memory:
section1: 900 -- 999 , size: 100 bytes.
$command:~free 100 bytes at address 0.
available memory:
section1: 900 -- 999 , size: 100 bytes.
section2: 0 -- 99 , size: 100 bytes.
$command:~alloc 20 bytes at address 900.
available memory:
section1: 920 -- 999 , size: 80 bytes.
section2: 0 -- 99 , size: 100 bytes.
$command:~free 100 bytes at address 700.
available memory:
section1: 920 -- 999 , size: 80 bytes.
section2: 0 -- 99 , size: 100 bytes.
section3: 700 -- 799 , size: 100 bytes.
$command:~free 100 bytes at address 300.
available memory:
section1: 920 -- 999 , size: 80 bytes.
section2: 0 -- 99 , size: 100 bytes.
section3: 300 -- 399 , size: 100 bytes.
section4: 700 -- 799 , size: 100 bytes.
$command:~free 100 bytes at address 400.
available memory:
section1: 920 -- 999 , size: 80 bytes.
section2: 0 -- 99 , size: 100 bytes.
section3: 300 -- 499 , size: 200 bytes.
section4: 700 -- 799 , size: 100 bytes.
$command:~free 100 bytes at address 600.
available memory:
section1: 920 -- 999 , size: 80 bytes.
section2: 0 -- 99 , size: 100 bytes.
section3: 300 -- 499 , size: 200 bytes.
section4: 600 -- 799 , size: 200 bytes.
$command:~free 100 bytes at address 500.
available memory:
section1: 920 -- 999 , size: 80 bytes.
section2: 0 -- 99 , size: 100 bytes.
section3: 300 -- 799 , size: 500 bytes.
$command:~quit.
free(): double free detected in tcache 2
Aborted (core dumped)
```

测试 2

样例

```
wzf@wzf:~/courses/OS$ cat test2.txt
m 900
f 100 100
f 100 0
f 300 400
f 0 400
f 1 399
q
```

输出

```
wzf@wzf:~/courses/OS$ ./main < test2.txt
$command:~alloc 900 bytes at address 0.
available memory:
section1: 900 -- 999 , size: 100 bytes.
$command:~free 100 bytes at address 100.
available memory:
section1: 900 -- 999 , size: 100 bytes.
section2: 100 -- 199 , size: 100 bytes.
$command:~min address 0 out of bound.
available memory:
section1: 900 -- 999 , size: 100 bytes.
section2: 100 -- 199 , size: 100 bytes.
$command:~free 300 bytes at address 400.
available memory:
section1: 900 -- 999 , size: 100 bytes.
section2: 100 -- 199 , size: 100 bytes.
section3: 400 -- 699 , size: 300 bytes.
$command:~min address 400 out of bound.
available memory:
section1: 900 -- 999 , size: 100 bytes.
section2: 100 -- 199 , size: 100 bytes.
section3: 400 -- 699 , size: 300 bytes.
$command:~min address 399 out of bound.
available memory:
section1: 900 -- 999 , size: 100 bytes.
section2: 100 -- 199 , size: 100 bytes.
section3: 400 -- 699 , size: 300 bytes.
$command:~quit.
free(): double free detected in tcache 2
Aborted (core dumped)
```

## 6. 经验与体会

- 经过此次实验，我发现由于不常用 c 语言编程，忘记了很多细节，诸如遗留在输入缓冲区的字符会造成 bug。
- 申请的空间一定要释放掉，在 c 语言为指针申请空间需要显示调用库函数，而在 c++ 中只需要 new，很容易忘记 delete。

- 在释放空间时需要考虑各种情况。不仅有目标区域与前后空闲区邻接关系的四种情况，还应该考虑目标区域越过最大范围，目标区域与其他空闲区重叠的情况。
- 程序最终的测试十分依赖于用例的构造，我在测试过程中不仅发现了仅有头节点一个节点时申请/释放空间造成的异常，甚至还遇到在输入几个特定的命令后出现死循环的情况。因此好的用例可以显著提高程序的健壮性。