

---

# 实验一      可变分区存储管理实验报告

---

## 1 程序设计要求

编写一个 C 程序，用 `char *malloc(unsigned size)` 函数向系统申请一块内存空间（如 `size = 1000`，单位为字节），用循环首次适应法 `addr = (char *)lmalloc(unsigned size)` 和 `lfree(unsigned size, char * addr)` 模拟 UNIX 可变分区内存管理，实现对该内存区的分配和释放管理。

## 2 实验目的

1. 加深对可变分区的存储管理的理解。
2. 提高用 C 语言编制大型系统程序的能力，特别是掌握 C 语言编程的难点：指针，以及将指针作为函数参数。
3. 掌握用指针实现链表和在链表上的基本操作。

## 3 算法思想

### 3.1 可变分区存储

可变分区存储不是预先把内存中的用户区域划分成若干固定分区，而是在作业要求装入内存时，根据用户作业的大小和当时内存空间使用情况决定是否为该作业分配一个分区以及分配分区的大小。因此分区大小不是预先固定的，分区的个数和位置也不是预先确定的。除此以外，可变分区存储不会像固定分区一样产生所谓的“内零头”，但是可能会产生“外零头”。

### 3.2 可变分区存储管理

空闲分区使用链表进行管理，将各个空闲块按一定顺序进行连接。在实际应用中，主要有以下几种管理方法：

1. **首次适配 (First Fit)：**将空闲分区链表中的空闲分区按照地址从低到高进行排序。算法从空闲分区表的第一个表项开始，查找最先能够满足要求的空闲分区。从此分区中划分出作业所需的大小分配给作业，剩余部分保留在空闲分区表中。

该算法倾向于优先利用内存中低地址部分的空闲分区，虽然保留了高地址部分的大块空闲区，且释放内存时操作简单，但是也可能导致低地址部分出现许多小空闲分区，增大每次查找时的开销。

2. **循环首次适配 (Next Fit):** 与首次适配算法类似，将空闲分区链表中的空闲分区按照地址从低到高进行排序。但是在分配内存时，不再每次从链表头开始查找，而是从上一次进行分配空闲区的下一个空闲分区开始查找，直至找到最先能够满足要求的空闲分区。从此分区中划分出作业所需的大小分配给作业，剩余部分保留在空闲分区表中。

该算法能使内存中的空闲分区分布得较为均匀，改善了首次适配算法查找开销过大的问题。

3. **最佳适配 (Best Fit):** 将空闲分区链表中的空闲分区按照空闲分区的大小从小到大排序。算法每次从空闲分区表的第一个表项开始，查找最先能够满足要求的空闲分区。从此分区中划分出作业所需的大小分配给作业，并将剩余部分重新插入空闲分区表中。

该算法能保存较大的空闲分区，并使分配后产生的碎片尽量小，但是仍会产生较多“外零头”，影响查找时的性能。

4. **最差适配 (Worst Fit):** 将空闲分区链表中的空闲分区按照空闲分区的大小从大到小排序。算法每次查看空闲分区表的第一个表项，若该分区大小满足要求，则从此分区中划分出作业所需的大小分配给作业，并将剩余部分重新插入空闲分区表中。若不能满足，则不进行分配。

该算法使得空闲分区链表中的表项大小趋于均匀，尽量降低小碎片产生的可能。

上述几种算法的实现效果如图1所示。

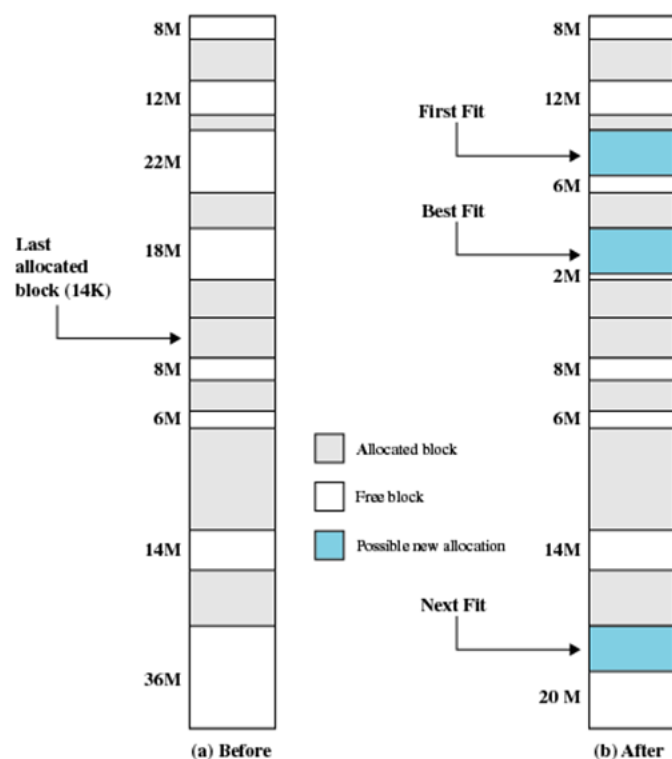


图 1: 几种可变分区存储管理法效果

### 3.3 循环首次适配算法

本次实验中进行内存分配时使用的是循环首次适应算法。

实验过程中，每一个空闲分区使用一个 map 结构进行管理，按照每个空闲分区的地址从低到高排列，整体形成一个双向链表。除此以外，设置一个全局变量指针，指向下一次分配时开始查找的空闲分区。

#### 3.3.1 分配过程

每次分配时从查找指针指向的空闲分区开始查找，直到找到第一个满足需求的空闲区。对该分区进行划分，分配作业所需的空间，即修改该分区的起始地址、大小等信息，若分区大小恰好等于作业所需内存大小，则还需要将该分区从链表中删除。这之后更改查找链表的值，使之指向分配分区的下一块空闲分区。

#### 3.3.2 释放过程

循环首次适配算法的释放过程与首次适配算法的类似，共有四种可能的情况：

1. 待释放区前面为空闲区，后面为已分配区，如图2(a)所示。释放时，空闲区与待释放区合并形成一个更大的空闲区。只需要将空闲区对应表项的大小修改为空闲区和待释放区大小之和即可。
2. 待释放区前后均为空闲区，如图2(b)所示。释放时，两个空闲区和待释放区合并形成一个更大的空闲区。此时需要修改第一个空闲区对应表项的大小为三个区域大小之和，并从空闲链表中删除第二块空闲区对应的表项。
3. 待释放区前面为已分配区，后面为空闲区，如图2(c)所示。释放时，待释放区和后面的空闲区合并形成一个更大的空闲区。需要将空闲区对应表项的起始地址修改为待释放区的起始地址，并将大小修改为两个区域大小之和。
4. 待释放区前后均为已分配区，如图2(d)所示。释放时，需要新建一个 map 结构存储释放后的空闲区表项，其起始地址为待释放区的起始地址，大小为待释放区的大小。然后将该表项插入空闲分区链表中合适的位置。

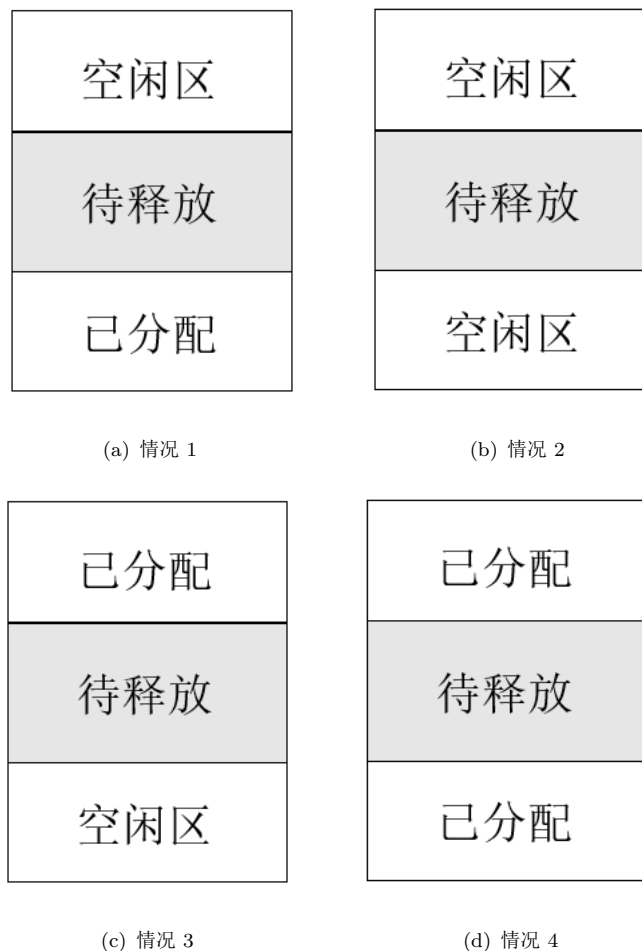


图 2: 释放内存时的四种情况

## 4 详细设计及功能

本次实验主要涉及了循环首次适配算法实现和测试两大部分。其中，循环首次适配算法的实现主要分为以下几个模块：

1. 输入处理模块。处理用户输入的命令，进行相应的操作，实现与用户的交互。
2. 内存分配、释放模块。算法的主体部分，实现了首次适配算法的分配及释放过程。
3. 打印模块。

### 4.1 接口说明

本小节将简要介绍每个模块中主要使用的函数接口，如下所示：

1. 初始化模块。

```
1  /*
2  ** to initiate the empty memory link
3  ** input:
4  **     baseaddr: ptr to the ptr to the base address of
5  **             meory
6  **     number: the number of maps of memory link
7  **     maphead: ptr to the ptr to the head map of empty
8  **             memory link
9  ** output: None
10 */
11 void initiatemap(char** baseaddr, int* number, Map** maphead
12 );
```

传入指定参数后，可以得到一块起始地址为 *baseaddr*，大小为 1000 字节的空闲内存块，该块的信息记录在 *\*maphead* 指向的结构体中。

2. 输入控制模块。

```
1  /*
2  ** switch the execute function accroding to the parameters
3  ** input by the user
4  ** input:
5  **     number: the total number of maps in the empty memory
6  **             link
7  **     search: the ptr to the Map that searching begins
8  **     head: the ptr to the first map of empty memory link
9  **     tocontinue: flag to record the quit operation
10 **     baseaddr: the beginning address of whole memory space
11 ** output: None
12 */
```

```

11 void control_input(int* number, Map** search, Map** head,
    int* tocontinue, const char* baseaddr);

```

根据用户的不同输入跳转执行相应功能，有效的输入包括：

- “m” 或 “M”：分配指定大小的内存。
- “f” 或 “F”：从指定地址开始释放指定大小的内存。
- “q” 或 “Q”：退出此程序。

### 3. 内存分配模块。

```

1  /*
2  ** allocate the memory according to the empty memory link
3  ** input:
4  **     size: the size of the memory to be allocated;
5  **     number: the total number of maps in the empty memory
        link;
6  **     search: ptr to the ptr to the Map that searching
        begins
7  ** output: None
8  */
9  void lmalloc(size_t size, int * number, Map **search);

```

从 \*search 指向的内存块开始搜索，找到合适大小的进行内存分配。

### 4. 内存释放模块。

```

1  /*
2  ** free an allocated memory according to the command
3  ** input:
4  **     size: the size of memory needed to free;
5  **     addr: beginning address of map;
6  **     head: ptr to the ptr to the first map of empty memory
        link;
7  **     number: the number of maps in the empty memory link;
8  **     baseaddr: the beginning address of whole memory space
        ;
9  ** output: result of free
10 **     1: success;
11 **     0: fail;
12 */
13 int lfree(size_t size, char * addr, Map** head, int *number,
    const char* baseaddr);

```

若输入合法，则释放起始地址为 addr、大小为 size 的内存块。

### 5. 打印模块。

```

1  /*
2  ** print message of a map in the empty memory link
3  ** input:
4  **     para: ptr to the map to be printed
5  ** output: None
6  */
7  void printmap(Map *para);
8
9  /*
10 ** print messages of all maps in the empty memory link
11 ** input:
12 **     head: ptr to the first map of empty memory link;
13 **     number: the number of maps in the empty memory link
14 ** output: None
15 */
16 void printmap_n(Map *head, int number);

```

使用 `printmap()` 函数可以打印出空闲链表中一个内存块的信息，而使用 `printmap_n()` 函数可以打印出空闲链表中全部内存块的信息。

## 5 重要数据结构及变量说明

本次实验过程中，主要使用的数据结构为保存空闲区信息的 **map** 结构体，包含：空闲区的大小、起始地址、两个分别指向前后空闲区的指针。

具体定义如下所示：

```

1  struct map{
2      unsigned m_size;
3      char* m_addr;
4      struct map *next, *prior;
5  };
6
7  typedef struct map Map;

```

随后定义了指向空闲内存链表头节点的指针 `maphead`，以及分配的 1000 字节内存的起始地址 `baseaddr`，如下所示：

```

1  char* baseaddr = NULL;
2  Map* maphead = NULL;


```

## 6 测试方法

在实验过程中，采用了如下两种测试方法。

## 6.1 交互式测试

用户通过控制台程序界面进行每次指令的输入。测试结果如图3、4、5所示：



```
D:\The Course Of SJTU\SJTU大三\大三下 操作系统实验一\memoryalloc\memoryalloc\x64\Debug\memoryalloc.exe
The base address of memory is: 00000236F7330570 .
=====
Initial state.
=====
-----
The message about this map of link is shown as follows.
The size is : 1000.
The beginning address is: 00000236F7330570.
The next map is: 00000236F7330570.
The prior map is: 00000236F7330570.
-----
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
m
Please input the size to be allocated.
NOTE: the size should be a integar.
10
-----
The message about this map of link is shown as follows.
The size is : 990.
The beginning address is: 00000236F733057A.
The next map is: 00000236F733057A.
The prior map is: 00000236F733057A.
-----
The alloc address next searching is: 00000236F733057A.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
m
Please input the size to be allocated.
NOTE: the size should be a integar.
200
-----
The message about this map of link is shown as follows.
The size is : 790.
The beginning address is: 00000236F7330642.
The next map is: 00000236F7330642.
The prior map is: 00000236F7330642.
-----
The alloc address next searching is: 00000236F7330642.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
f
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in [0, 1000]) is OK.
20
Please input the size of freed memory.
NOTE: the size should be a integar.
70
=====
Free succeeded!
=====
-----
The message about this map of link is shown as follows.
The size is : 70.
The beginning address is: 00000236F7330584.
The next map is: 00000236F7330642.
The prior map is: 00000236F7330642.
-----
The message about this map of link is shown as follows.
```

图 3: 交互式测试结果 (1)



```

The size is : 790.
The beginning address is: 00000236F7330642.
The next map is: 00000236F7330584.
The prior map is: 00000236F7330584.
-----
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
f
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in [0, 1000]) is OK.
100
Please input the size of freed memory.
NOTE: the size should be a integer.
64
=====
Free succeeded!
=====
-----
The message about this map of link is shown as follows.
The size is : 70.
The beginning address is: 00000236F7330584.
The next map is: 00000236F73305D4.
The prior map is: 00000236F7330642.
-----
The message about this map of link is shown as follows.
The size is : 64.
The beginning address is: 00000236F73305D4.
The next map is: 00000236F7330642.
The prior map is: 00000236F7330584.
-----
The message about this map of link is shown as follows.
The size is : 790.
The beginning address is: 00000236F7330642.
The next map is: 00000236F7330584.
The prior map is: 00000236F73305D4.
-----
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
m
Please input the size to be allocated.
NOTE: the size should be a integer.
80
-----
The message about this map of link is shown as follows.
The size is : 70.
The beginning address is: 00000236F7330584.
The next map is: 00000236F73305D4.
The prior map is: 00000236F7330692.
-----
The message about this map of link is shown as follows.
The size is : 64.
The beginning address is: 00000236F73305D4.
The next map is: 00000236F7330692.
The prior map is: 00000236F7330584.
-----

```

图 4: 交互式测试结果 (2)

```

-----
The message about this map of link is shown as follows.
The size is : 710.
The beginning address is: 00000236F7330692.
The next map is: 00000236F7330584.
The prior map is: 00000236F73305D4.
-----
The alloc address next searching is: 00000236F7330584.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
q
Having free the 1 th map.
Having free the 2 th map.
Having free the 3 th map.
请按任意键继续. . .

```

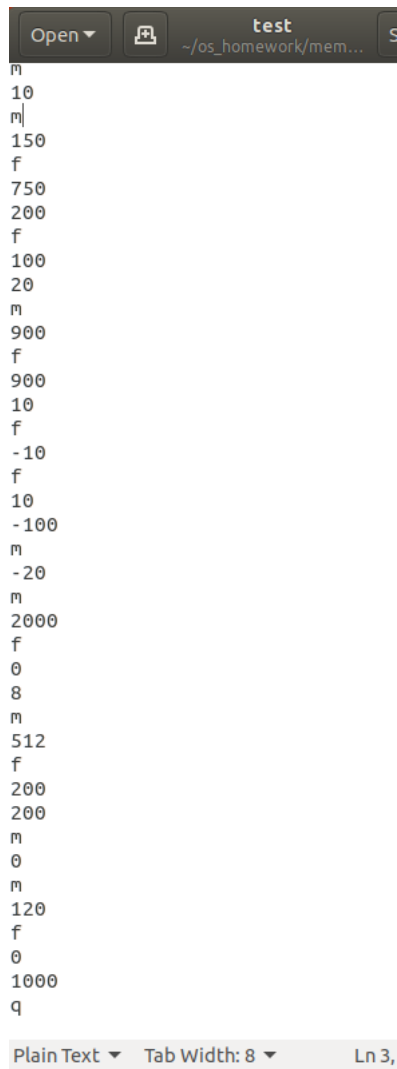
图 5: 交互式测试结果 (3)

## 6.2 文件测试

在 Linux 下使用重定向实现。命令为: `./memory < test > result`, 即将 test 文件重定向到标准输入中, 再将标准输出重定向到 result 文件。

测试时使用的 test 文件内容如图6所示。

具体流程为: 1、分配 10 字节大小内存。2、分配 150 字节大小内存。3、释放起始地址为 750、大小为 200 字节的内存。4、释放起始地址为 100、大小为 20 字节的内存。5、分配 900 字节内存。6、释放起始地址为 900、大小为 10 字节的内存。7、释放起始地址为-10 的内存块 (此时程序不需要再读入第二个有关内存大小的参数)。8、释放起始地址为 10、大小为-100 字节的内存。9、分配-20 字节内存。10、分配 2000 字节内存。11、释放起始地址为 0、大小为 8 字节的内存。12、分配 512 字节内存。13、释放起始地址为 200、大小为 200 字节的内存。14、分配 0 字节内存。15、分配 120 字节内存。16、释放起始地址为 0、大小为 1000 字节的内存。



```
m
10
m
150
f
750
200
f
100
20
m
900
f
900
10
f
-10
f
10
-100
m
-20
m
2000
f
0
8
m
512
f
200
200
m
0
m
120
f
0
1000
q
```

Plain Text ▾ Tab Width: 8 ▾ Ln 3,

图 6: 测试文件内容

最终测试结果输出至 result 文件，如图8所示。由于输出结果较长，因此在这里仅放置第一张输出结果，其余输出结果见附录。



```
Open  result  Save  ~/os_homework/memoryalloc
The base address of memory is: 0x55f05d391260 .
=====
Initial state.
=====
-----
The message about this map of link is shown as follows.
The size is : 1000.
The beginning address is: 0x55f05d391260.
The next map is: 0x55f05d391260.
The prior map is: 0x55f05d391260.
-----
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please input the size to be allocated.
NOTE: the size should be a integar.
-----
The message about this map of link is shown as follows.
The size is : 990.
The beginning address is: 0x55f05d39126a.
The next map is: 0x55f05d39126a.
The prior map is: 0x55f05d39126a.
-----
The alloc address next searching is: 0x55f05d39126a.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please input the size to be allocated.
NOTE: the size should be a integar.
-----
The message about this map of link is shown as follows.
The size is : 840.
The beginning address is: 0x55f05d391300.
The next map is: 0x55f05d391300.
The prior map is: 0x55f05d391300.
-----
The alloc address next searching is: 0x55f05d391300.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in
[0, 1000]) is OK.
Please input the size of freed memory.
Plain Text  Tab Width: 8  Ln 1, Col 1  INS
```

图 7: 测试结果 (1)

### 6.3 编译环境

本次实验所用代码在 Microsoft Visual Studio 2017 和 Ubuntu 18.04 gcc 7.4.0 上均经过测试。

## 7 程序及测试的改进

- 进一步完善了释放内存时对各种不能执行释放的情况的考虑，如待释放区域不能与空闲区有交集等，增强了程序的鲁棒性。
- 增加了更多对于异常内存释放的测试语句，如：待释放区与空闲区有部分重叠、待释放区完整地包含空闲区。

- 在用户输入错误指令后，提供更为详细的提示内容，如：“输入的内存大小必须为正整数”等，优化了交互界面。

## 8 遇到的问题

在本次实验过程中，我也遇到了一些问题：

1. 指针的概念及使用。指针作为 C 语言较为重要同时也是较有难度的一个知识点，一直以来我都掌握得不太好。通过本次实验过程，我进一步加深了对指针概念的理解。指针指向的只是一块内存空间，具体这块空间中的数据代表什么意义，与指向它的指针的类型有关。除此之外 C 语言在调用函数时，对各个参数都是进行值传递，指针也不例外。因此如果需要在函数中修改指针指向的对象，需要使用指向指针的指针，即二级指针。
2. 从缓冲区读取数据。使用 `scanf()` 函数从缓冲区读取用户输入的数据时，还需要处理残留在缓冲区中的换行符 `'\n'`，否则它会被 `scanf()` 当作下一个输入进行读取，影响之后程序的运行。因此在每次读取数据之后，需要清空缓冲区。常用的清空缓冲区的方法有：

- `fflush(stdin)`。但是在 C 和 C++ 标准中从未定义过此函数，是对 C 标准的扩充，因此并非所有的编译器都支持这个功能。
- `while()` 循环。使用如下两种语句读取掉缓冲区中的剩余字符。

– a)

```
1 while((ch = getchar()) != EOF && ch != '\n')
2     ;
```

– b)

```
1 do
2     c=getchar();
3 while(c == '\n' || c == '\t' || c == ' ');
```

3. 释放操作时对是否可以释放的判断。循环首次适配算法相比其他算法在释放时较为简单，但是仍需要分四种情况进行考虑。除此之外，还有一些属于错误释放的输入，应对这些异常输入进行处理：
  - 要释放的地址空间超过规定的空间大小。
  - 要释放的起始地址不处于规定的地址空间内。
  - 要释放的空间全部或部分位于空闲区。
  - 要释放的空间大小为负数。
4. 32 位与 64 位移植性问题。由于在 Windows 中使用的是 32 位平台进行编译的，而我使用的 Ubuntu 为 64 位系统，使用 gcc 编译时默认生成 64 位的应用程

序。由于代码中使用了 *unsigned int* 数据类型，因此运行时出现了地址被截断的现象。

该问题有两种解决方法：

- (1) 使用 gcc 进行编译时添加参数 “-m32”，使之生成 32 位应用程序。
- (2) 修改原代码，使用 “\_\_int64” 或 “\_\_int32” 等数据类型，使程序适应 64 位系统。

## 9 心得体会

通过本次实验，我对于可变分区存储管理的多种机制有了更为全面的了解，特别是对于循环首次适配算法有了更为清晰的认识，巩固了课堂所学。对于异常内存释放的情况进行了较为全面的考量，培养了细致思考的能力。除此以外，在完成实验的过程中，对于 C 语言中指针的概念有了进一步的理解，也能够较为熟练地进行使用。在将程序移植到 Windows 和 Linux 系统的过程中，认识到了 64 位和 32 位系统对于编程的影响，知道了使用数据类型时应该考虑可移植性，很好地锻炼了我们的专业技能。在实验过程中虽然遇到了不少困难，但是通过与同学、老师进行交流、上网查阅资料等途径，较好地解决了它们，培养了协作精神。

总之，通过本次实验我收获颇丰，衷心感谢刘老师在实验过程中的指导与帮助！

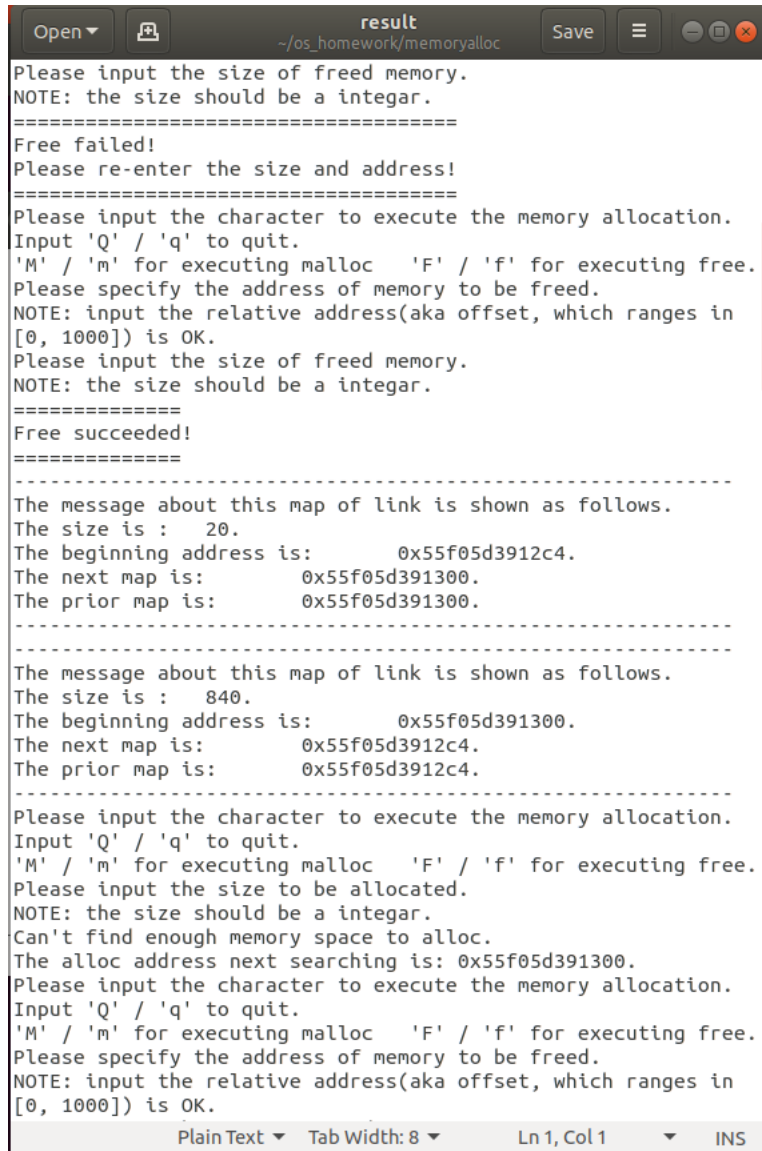
## 10 附录

### 10.1 测试结果展示

使用 test 文件进行测试的全部结果如图8、9、10、11、12、13、14所示。

```
Open ▾  result  Save  ≡  -/os_homework/memoryalloc
The base address of memory is: 0x55f05d391260 .
=====
Initial state.
=====
-----
The message about this map of link is shown as follows.
The size is : 1000.
The beginning address is: 0x55f05d391260.
The next map is: 0x55f05d391260.
The prior map is: 0x55f05d391260.
-----
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please input the size to be allocated.
NOTE: the size should be a integar.
-----
The message about this map of link is shown as follows.
The size is : 990.
The beginning address is: 0x55f05d39126a.
The next map is: 0x55f05d39126a.
The prior map is: 0x55f05d39126a.
-----
The alloc address next searching is: 0x55f05d39126a.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please input the size to be allocated.
NOTE: the size should be a integar.
-----
The message about this map of link is shown as follows.
The size is : 840.
The beginning address is: 0x55f05d391300.
The next map is: 0x55f05d391300.
The prior map is: 0x55f05d391300.
-----
The alloc address next searching is: 0x55f05d391300.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in
[0, 1000]) is OK.
Please input the size of freed memory.
Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS
```

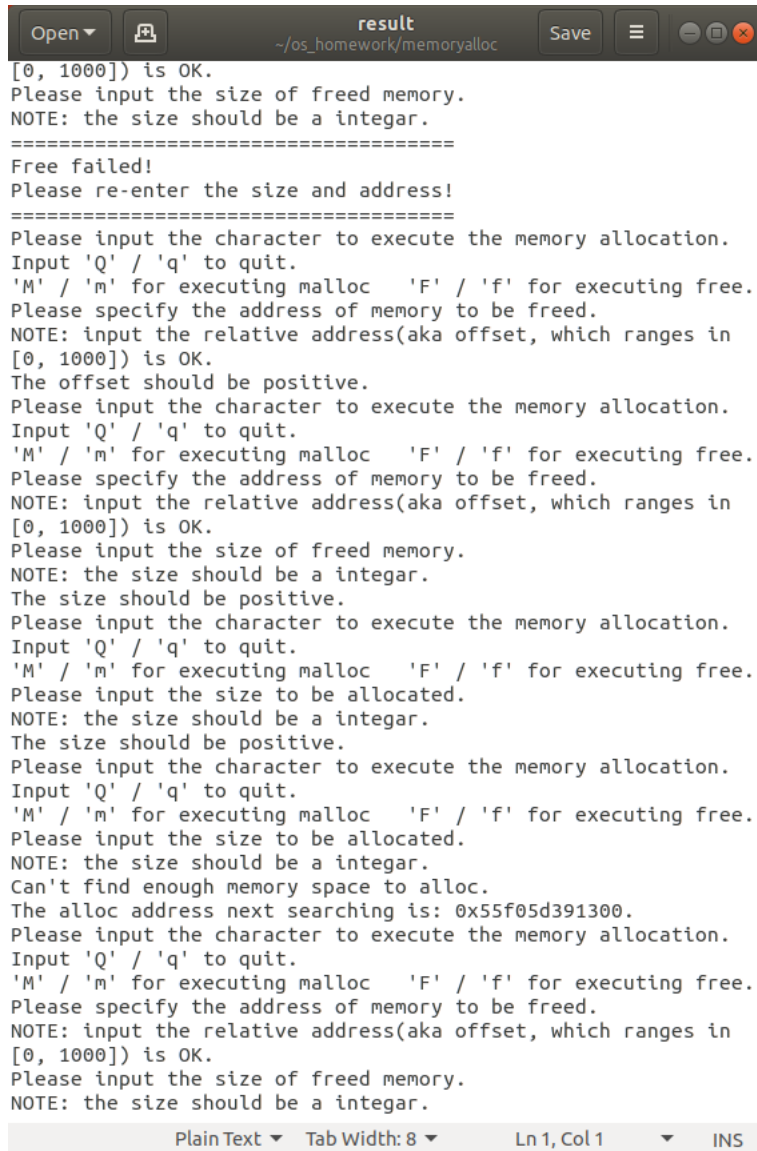
图 8: 测试结果 (1)



```
Open  result  Save  ~/.os_homework/memoryalloc
Please input the size of freed memory.
NOTE: the size should be a integar.
=====
Free failed!
Please re-enter the size and address!
=====
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc  'F' / 'f' for executing free.
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in
[0, 1000]) is OK.
Please input the size of freed memory.
NOTE: the size should be a integar.
=====
Free succeeded!
=====
-----
The message about this map of link is shown as follows.
The size is : 20.
The beginning address is: 0x55f05d3912c4.
The next map is: 0x55f05d391300.
The prior map is: 0x55f05d391300.
-----
The message about this map of link is shown as follows.
The size is : 840.
The beginning address is: 0x55f05d391300.
The next map is: 0x55f05d3912c4.
The prior map is: 0x55f05d3912c4.
-----
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc  'F' / 'f' for executing free.
Please input the size to be allocated.
NOTE: the size should be a integar.
Can't find enough memory space to alloc.
The alloc address next searching is: 0x55f05d391300.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc  'F' / 'f' for executing free.
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in
[0, 1000]) is OK.
Plain Text  Tab Width: 8  Ln 1, Col 1  INS
```

图 9: 测试结果 (2)





```
Open ▾  result  Save  ~ /os_homework/memoryalloc
[0, 1000]) is OK.
Please input the size of freed memory.
NOTE: the size should be a integar.
=====
Free failed!
Please re-enter the size and address!
=====
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc  'F' / 'f' for executing free.
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in
[0, 1000]) is OK.
The offset should be positive.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc  'F' / 'f' for executing free.
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in
[0, 1000]) is OK.
Please input the size of freed memory.
NOTE: the size should be a integar.
The size should be positive.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc  'F' / 'f' for executing free.
Please input the size to be allocated.
NOTE: the size should be a integar.
The size should be positive.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc  'F' / 'f' for executing free.
Please input the size to be allocated.
NOTE: the size should be a integar.
Can't find enough memory space to alloc.
The alloc address next searching is: 0x55f05d391300.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc  'F' / 'f' for executing free.
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in
[0, 1000]) is OK.
Please input the size of freed memory.
NOTE: the size should be a integar.
```

Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

图 10: 测试结果 (3)

```
Open ▾ [icon] *result ~/os_homework/memoryalloc Save [icon] [icon] [icon] [icon]
NOTE: the size should be a integar.
=====
Free succeeded!
=====
-----
The message about this map of link is shown as follows.
The size is : 8.
The beginning address is: 0x55f05d391260.
The next map is: 0x55f05d3912c4.
The prior map is: 0x55f05d391300.
-----
-----
The message about this map of link is shown as follows.
The size is : 20.
The beginning address is: 0x55f05d3912c4.
The next map is: 0x55f05d391300.
The prior map is: 0x55f05d391260.
-----
-----
The message about this map of link is shown as follows.
The size is : 840.
The beginning address is: 0x55f05d391300.
The next map is: 0x55f05d391260.
The prior map is: 0x55f05d3912c4.
-----
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please input the size to be allocated.
NOTE: the size should be a integar.
-----
The message about this map of link is shown as follows.
The size is : 8.
The beginning address is: 0x55f05d391260.
The next map is: 0x55f05d3912c4.
The prior map is: 0x55f05d391500.
-----
-----
The message about this map of link is shown as follows.
The size is : 20.
The beginning address is: 0x55f05d3912c4.
The next map is: 0x55f05d391500.
The prior map is: 0x55f05d391260.
-----
Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 2 ▾ INS
```

图 11: 测试结果 (4)

```
Open  *result  Save  ~ /os_homework/memoryalloc
-----
The message about this map of link is shown as follows.
The size is : 328.
The beginning address is: 0x55f05d391500.
The next map is: 0x55f05d391260.
The prior map is: 0x55f05d3912c4.
-----
The alloc address next searching is: 0x55f05d391260.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in
[0, 1000]) is OK.
Please input the size of freed memory.
NOTE: the size should be a integar.
=====
Free succeeded!
=====
-----
The message about this map of link is shown as follows.
The size is : 8.
The beginning address is: 0x55f05d391260.
The next map is: 0x55f05d3912c4.
The prior map is: 0x55f05d391500.
-----
The message about this map of link is shown as follows.
The size is : 20.
The beginning address is: 0x55f05d3912c4.
The next map is: 0x55f05d391328.
The prior map is: 0x55f05d391260.
-----
The message about this map of link is shown as follows.
The size is : 200.
The beginning address is: 0x55f05d391328.
The next map is: 0x55f05d391500.
The prior map is: 0x55f05d3912c4.
-----
The message about this map of link is shown as follows.
The size is : 328.
The beginning address is: 0x55f05d391500.
Plain Text  Tab Width: 8  Ln 1, Col 2  INS
```

图 12: 测试结果 (5)

```
Open ▾ [icon] *result ~/os_homework/memoryalloc Save [icon] [icon] [icon] [icon]
The message about this map of link is shown as follows.
The size is : 328.
The beginning address is: 0x55f05d391500.
The next map is: 0x55f05d391260.
The prior map is: 0x55f05d391328.
-----
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please input the size to be allocated.
NOTE: the size should be a integar.
The size should be positive.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please input the size to be allocated.
NOTE: the size should be a integar.
-----
The message about this map of link is shown as follows.
The size is : 8.
The beginning address is: 0x55f05d391260.
The next map is: 0x55f05d3912c4.
The prior map is: 0x55f05d391500.
-----
The message about this map of link is shown as follows.
The size is : 20.
The beginning address is: 0x55f05d3912c4.
The next map is: 0x55f05d3913a0.
The prior map is: 0x55f05d391260.
-----
The message about this map of link is shown as follows.
The size is : 80.
The beginning address is: 0x55f05d3913a0.
The next map is: 0x55f05d391500.
The prior map is: 0x55f05d3912c4.
-----
The message about this map of link is shown as follows.
The size is : 328.
The beginning address is: 0x55f05d391500.
The next map is: 0x55f05d391260.
The prior map is: 0x55f05d3913a0.
-----
Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 2 ▾ INS
```

图 13: 测试结果 (6)

```

-----
The alloc address next searching is: 0x55f05d391500.
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Please specify the address of memory to be freed.
NOTE: input the relative address(aka offset, which ranges in
[0, 1000]) is OK.
Please input the size of freed memory.
NOTE: the size should be a integar.
=====
Free failed!
Please re-enter the size and address!
=====
Please input the character to execute the memory allocation.
Input 'Q' / 'q' to quit.
'M' / 'm' for executing malloc 'F' / 'f' for executing free.
Having free the 1 th map.
Having free the 2 th map.
Having free the 3 th map.
Having free the 4 th map.
Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 2 ▾ INS

```

图 14: 测试结果 (7)

## 10.2 源程序代码

实验过程中的源程序代码如下所示:

```

1 // ma.h
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #define MAX_MEMORY 1000
6
7 struct map{
8     unsigned m_size;
9     char* m_addr;
10    struct map *next, *prior;
11 };
12
13 typedef struct map Map;
14
15 /*
16  ** to initiate the empty memory link
17  ** input:
18  **     baseaddr: ptr to the base address of memeory
19  **     number: the number of maps of memory link
20  ** output:
21  **     head: ptr to the first map of empty memory link
22  */
23 void initiatemap(char** baseaddr, int *number, Map** maphead)
24 {
25     *baseaddr = (char*) malloc(MAX_MEMORY);

```

```

26     if (*baseaddr != NULL)
27     {
28         printf("The base address of memory is: %p\t.\n", *
                baseaddr);
29     }
30     else {
31         printf("malloc failed!\n");
32         exit(1);
33     }
34     *maphead = (Map*) malloc(sizeof(Map));
35     if (*maphead == NULL)
36         printf("Error");
37     (*maphead)->m_addr = *baseaddr;
38     (*maphead)->m_size = 1000;
39     (*maphead)->next = *maphead;
40     (*maphead)->prior = *maphead;
41     (*number)++;
42 };
43
44 /*
45 ** to initiate the memory space with 4 maps, structure is shown
    below
46 ** 128(empty) || 22(occupied) || 256(empty) || 44(occupied) || \
47    128(empty) || 22(occupied) || 128(empty) || 272(occupied)
48 ** input:
49 **     baseaddr: ptr to the base address of memeory
50 **     number: the number of maps of memory link
51 ** output:
52 **     head: ptr to the first map of empty memory link
53 */
54 void random_insert(Map* head, int *number, char* baseaddr)
55 {
56     // Map *new_m1 = (Map*) malloc(sizeof(Map));
57     Map *new_m2 = (Map*) malloc(sizeof(Map));
58     Map *new_m3 = (Map*) malloc(sizeof(Map));
59     Map *new_m4 = (Map*) malloc(sizeof(Map));
60
61     head->m_size = 128;
62     head->m_addr = baseaddr;
63     head->next = new_m2;
64     head->prior = new_m4;
65
66     //new_m1->m_size = 128;
67     //new_m1->m_addr = baseaddr;
68     //new_m1->next = new_m2;
69     //new_m1->prior = new_m4;

```

```

70     // head = new_m1;
71     // (*number)++;
72
73     new_m2->m_size = 256;
74     new_m2->m_addr = (char*)(baseaddr + 150);
75     new_m2->next = new_m3;
76     new_m2->prior = head;
77     (*number)++;
78
79     new_m3->m_size = 128;
80     new_m3->m_addr = (char*)(baseaddr + 450);
81     new_m3->next = new_m4;
82     new_m3->prior = new_m2;
83     (*number)++;
84
85     new_m4->m_size = 128;
86     new_m4->m_addr = (char*)(baseaddr + 600);
87     new_m4->next = head;
88     new_m4->prior = new_m3;
89     (*number)++;
90
91     // return new_m1;
92 };
93
94 /*
95  ** print a map in the empty memory link
96  ** input:
97  **     para: ptr to the map to be printed
98  ** output: None
99  */
100 void printmap(Map *para)
101 {
102     printf("—————\n");
103     printf("The message about this map of link is shown as\n");
104         follows.\n");
105     printf("The size is : \t%d.\n", para->m_size);
106     printf("The beginning address is : \t%p.\n", para->m_addr);
107     printf("The next map is : \t%p.\n", (para->next)->m_addr);
108     printf("The prior map is : \t%p.\n", (para->prior)->m_addr);
109     printf("—————\n");
110 };
111
112 /*
113  ** print all maps in the empty memory link
114  ** input:
115  **     head: ptr to the first map of empty memory link;

```

```

115  **      number: the number of maps in the empty memory link
116  ** output: None
117  */
118  void printmap_n(Map *head, int number)
119  {
120      for (int i = 0; i < number; i++)
121      {
122          printmap(head);
123          head = head->next;
124      }
125  };
126
127  /*
128  ** remove a map from the empty memory link
129  ** input:
130  **      p: previous map of the map will be removed;
131  **      c: the map to be removed;
132  **      n: the next map of the map will be removed;
133  ** output: None
134  */
135  void remove_map(Map *p, Map *c, Map *n)
136  {
137      p->next = n;
138      n->prior = p;
139      free(c);
140  };
141
142  /*
143  ** find the map that has lowest address
144  ** input:
145  **      begin: the current head of empty memory link
146  **      number: the number of maps in the empty memory link
147  ** output:
148  **      ptr: the map should be the head of empty memory link
149  */
150  Map* findthelowest(Map* begin, int number)
151  {
152      Map* ptr = begin;
153      for (int i = 0; i < number; ++i)
154      {
155          if (ptr->m_addr > begin->m_addr)
156              ptr = begin;
157          begin = begin->next;
158      }
159      return ptr;
160  };

```



```

161
162 /*
163 ** allocate the memory according to the empty memory link
164 ** input:
165 **     size: the size of the memory to be allocated;
166 **     number: the total number of maps in the empty memory link;
167 **     search: ptr to the ptr to the Map that searching begins
168 ** output: None
169 */
170 void lmalloc(unsigned size, int * number, Map **search)
171 {
172     int flag = 0;
173     Map *head = *search;
174     // Map *target = (Map *)malloc(sizeof(Map));
175     do
176     {
177         if ((*search)->m_size > size)
178         {
179             flag = 1;
180             (*search)->m_size -= size;
181             (*search)->m_addr = (char*)(*search)->
182                 m_addr + size);
183             (*search) = (*search)->next;
184             // target->prior = (*e_map)->prior;
185             // target->next = NULL;
186             break;
187         }
188         else
189         {
190             if ((*search)->m_size == size)
191             {
192                 flag = 1;
193                 (*number)--;
194                 if (*number)
195                 {
196                     Map* tmp = (*search)->next
197                         ;
198                     remove_map((*search)->
199                         prior, (*search), (*
200                         search)->next);
201                     (*search) = tmp;
202                 }
203                 break;
204             }
205             else
206                 continue;

```

```

203         }
204     } while ((*search) = (*search)->next, (*search)->next !=
        head);
205
206     if (flag)
207     {
208         if (!(*number))
209         {
210             printf("All memory has been allocated.\n");
211             (*search) = NULL;
212         }
213         else {
214             head = findthelowest(*search, *number);
215             printmap_n(head, *number);
216         }
217     }
218     else
219     {
220         (*search) = head;
221         printf("Can't find enough memory space to alloc.\n
            ");
222     }
223     printf("The alloc address next searching is: %p.\n", (*
        search)->m_addr);
224 };
225
226 /*
227 ** insert a new map to the empty memory link between lower and upper
228 ** input:
229 **     anew: the new map to be inserted
230 **     lower: the previous map of the new map
231 **     upper: the next map of the new map
232 ** output: None
233 */
234 void insertmap(Map* anew, Map* lower, Map* upper)
235 {
236     lower->next = anew;
237     anew->prior = lower;
238     anew->next = upper;
239     upper->prior = anew;
240 };
241
242 // input maybe not the beginning address of a map
243 /*
244 ** free an allocated memory according to the command
245 ** input:

```

```

246  **      size: the size of memory needed to free;
247  **      addr: beginning address of map;
248  **      head: ptr to the ptr to the first map of empty memory link;
249  **      number: the number of maps in the empty memory link;
250  **      baseaddr: the beginning address of whole memory space;
251  ** output: result of free
252  **      1: success;
253  **      0: fail;
254  */
255  int lfree(unsigned size, char * addr, Map** head, int *number,
           const char* baseaddr)
256  {
257      char * freebegin = addr;
258      char * freeend = (char *) (addr + size);
259      Map * ptr = *head;
260      // upper : the next empty map
261      Map* upper = NULL;
262      // lower : the previous empty map
263      Map* lower = NULL;
264      if (freeend > (baseaddr + MAX_MEMORY))
265          return 0;
266      if (freebegin < (baseaddr))
267          return 0;
268      if ((*head) != NULL)
269      {
270          do
271          {
272              if ((ptr->m_addr + ptr->m_size) <=
                  freebegin)
273              {
274                  lower = ptr;
275              }
276              if ((ptr->m_addr >= freeend) && upper ==
                  NULL)
277              {
278                  upper = ptr;
279              }
280          } while (ptr = ptr->next, ptr != (*head));
281          if (upper == NULL && lower == NULL)
282              return 0;
283      }
284      else
285      {
286          Map * new_map = (Map*) malloc(sizeof(Map));
287          new_map->m_addr = freebegin;
288          new_map->m_size = size;

```

```

289         new_map->next = new_map;
290         new_map->prior = new_map;
291         *head = new_map;
292         return 1;
293     }
294
295     // test the size is correct or not
296     if (lower != NULL || upper != NULL)
297     {
298         if ( lower != NULL && ( ((lower->next)->m_addr <
299             freebegin) && (lower->m_addr + lower->m_size)
300             > freebegin) )
301             return 0;
302         if (lower == NULL && (((*head)->m_addr < freebegin
303             ) && ((*head)->m_addr + (*head)->m_size >
304             freebegin)))
305             return 0;
306         if ((upper != NULL) && ( ((upper->prior)->m_addr <
307             freeend) && ((upper->prior)->m_addr + ((upper
308             ->prior)->m_size) > freeend) ) )
309             return 0;
310         if ((upper == NULL) && ((((*head)->prior)->m_addr
311             < freeend) && (((*head)->prior)->m_addr + ((*
312             head)->prior)->m_size > freeend)))
313             return 0;
314     }
315
316     if ( (lower != NULL) && ((lower->m_addr + lower->m_size)
317         == freebegin) )
318     {
319         /* empty || free || empty */
320         if ( (upper != NULL) && ((upper->next)->m_addr ==
321             freeend) )
322         {
323             lower->m_size += size;
324             (*number)--;
325             if ((*number) > 1)
326                 remove_map(lower, upper, upper->
327                     next);
328             else
329                 printf("The size of memory is 1000
330                     now.\n");
331         }
332         /* empty || free || allocation || empty(or end) */
333         else
334         {

```

```

323         lower->m_size += size;
324     }
325 }
326 /* empty (or begin) || allocation || free || ... */
327 else
328 {
329     /* empty (or begin) || allocation || free || empty */
330     if ( (upper != NULL) && (upper->m_addr == freeend) )
331     {
332         upper->m_addr = freebegin;
333         upper->m_size += size;
334     }
335     /* empty (or begin) || allocation || free || allocation || empty(or end) */
336     else
337     {
338         Map * new_map = (Map*) malloc(sizeof(Map));
339         new_map->m_addr = freebegin;
340         new_map->m_size = size;
341
342         /* empty (or begin) || allocation || free || allocation || empty */
343         if (upper != NULL)
344         {
345             insertmap(new_map, upper->prior,
346                     upper);
347             (*number)++;
348         }
349         /* empty (or begin) || allocation || free || allocation (end) */
350         else
351         {
352             if (lower != NULL)
353             {
354                 insertmap(new_map, lower,
355                         lower->next);
356                 (*number)++;
357             }
358         }
359     }
360     // changeheader(head, *number);
361     (*head) = findthelowest(*head, *number);
362     return 1;

```

```

362 };
363
364 /*
365  ** free the Map variables in the empty memory link
366  ** input:
367  **     head: the first map of the empty memory link
368  **     number: the total number of maps in the empty memory link
369  ** output: None
370  */
371 void freeallmap(Map *head, int number)
372 {
373     Map* ptr = head;
374     for (int i = 0; i < number; i++)
375     {
376         ptr = head->next;
377         free(head);
378         printf("Having free the %d th map.\n", i + 1);
379         head = ptr;
380     }
381 };
382
383 /*
384  ** switch the execute function according to the parameters input
385  ** by the user
386  ** input:
387  **     number: the total number of maps in the empty memory link
388  **     search: the ptr to the Map that searching begins
389  **     head: the ptr to the first map of empty memory link
390  **     tocontinue: flag to record the quit operation
391  **     baseaddr: the beginning address of whole memory space
392  ** output: None
393  */
394 void control_input(int* number, Map** search, Map** head, int*
395 tocontinue, const char* baseaddr)
396 {
397     printf("Please input the character to execute the memory
398 allocation.\n");
399     printf("Input 'Q' / 'q' to quit.\n");
400     printf("'M' / 'm' for executing malloc\t 'F' / 'f' for
401 executing free.\n");
402     char ch = '\0';
403     char para = '\0';
404     scanf("%c", &para);
405     while ((ch = getchar()) != EOF && ch != '\n')
406     ;
407     int size = 0, offset = 0;

```

```

404 char* ptr = NULL;
405 switch (para)
406 {
407     case 'm' : case 'M':
408         printf("Please input the size to be allocated.\n");
409         ;
410         printf("NOTE: the size should be a integar.\n");
411         scanf("%d", &size);
412         while ((ch = getchar()) != EOF && ch != '\n')
413             ;
414         if (size <= 0)
415         {
416             // *tocontinue = 0;
417             printf("The size should be positive.\n");
418             return;
419         }
420         lmalloc(size, number, search);
421         // printf("Now the search begins from: %p\n",
422             search->m_addr);
423         break;
424     case 'f' : case 'F':
425         printf("Please specify the address of memory to be
426             freed.\n");
427         // printf("NOTE: the address should be in HEX.\n");
428         ;
429         printf("NOTE: input the relative address(aka
430             offset, which ranges in [0, 1000]) is OK.\n");
431         scanf("%d", &offset);
432         while ((ch = getchar()) != EOF && ch != '\n')
433             ;
434         if (offset < 0)
435         {
436             printf("The offset should be positive.\n");
437             ;
438             // *tocontinue = 0;
439             return;
440         }
441         ptr = (char*)(baseaddr + offset);
442         // scanf("%p", &ptr);
443
444         printf("Please input the size of freed memory.\n");
445         ;
446         printf("NOTE: the size should be a integar.\n");
447         scanf("%d", &size);
448         while ((ch = getchar()) != EOF && ch != '\n')
449             ;

```

```

443         if (size <= 0 )
444         {
445             printf("The size should be positive.\n");
446             // *tocontinue = 0;
447             return;
448         }
449         if(lfree(size , ptr , head , number , baseaddr))
450         {
451             printf("=====\n");
452             printf("Free succeeded!\n");
453             printf("=====\n");
454             printmap_n(*head , *number);
455         }
456         else
457         {
458             printf("=====\n");
459             printf("Free failed!\nPlease re-enter the
              size and address!\n");
460             printf("=====\n");
461         }
462         break;
463     case 'q': case 'Q':
464         //while ((ch = getchar()) != EOF && ch != '\n')
465         //    ;
466         *tocontinue = 0;
467         break;
468     default:
469         printf("Please enter the correct key word!\n");
470     };
471 };

```

```

1  // m.c
2  #include "ma.h"
3
4  Map * next_search = NULL;
5
6  int main() {
7      int number = 0;
8      char ch = '\0';
9      char* baseaddr = NULL; // (char*) malloc(MAX_MEMORY);
10     Map* maphead = NULL;
11     int flagcontinue = 1;
12
13     initiatemap(&baseaddr, &number, &maphead);
14     // random_insert(maphead, &number, baseaddr);
15

```



```

16  next_search = maphead;
17
18  printf( "====\n" );
19  printf( "Initial state.\n" );
20  printf( "====\n" );
21  printmap_n(maphead, number);
22
23  while (1)
24  {
25      if (!flagcontinue)
26          break;
27      control_input(&number, &next_search, &maphead, &flagcontinue,
28                  baseaddr);
29  }
30
31  free(baseaddr);
32  freeallmap(maphead, number);
33  baseaddr = NULL;
34  system( "pause" );
35  return 0;

```