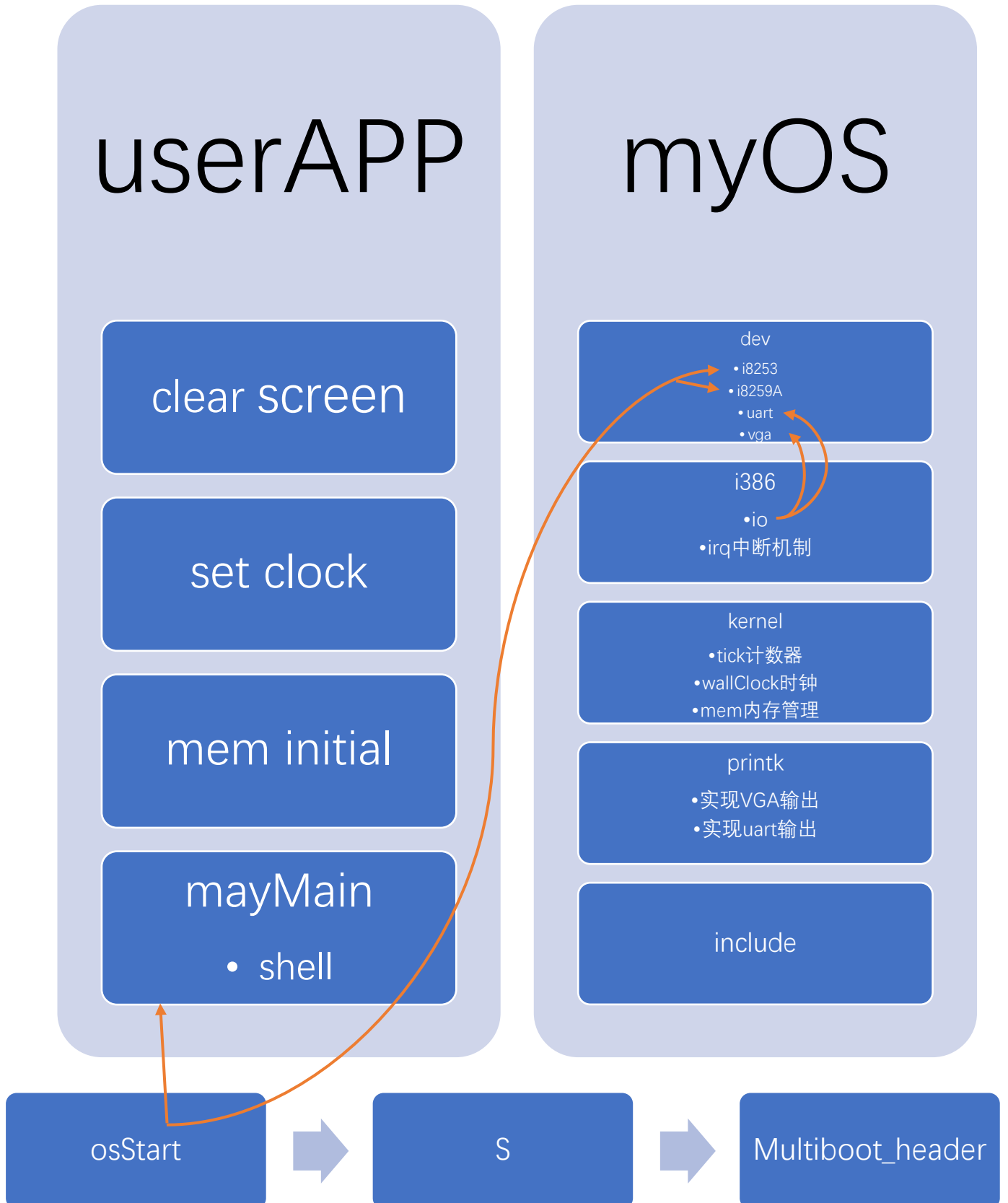


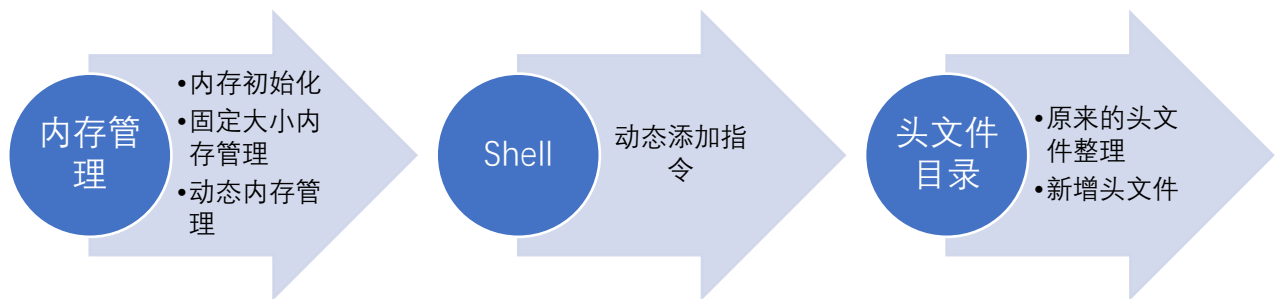
操作系统实验 4 报告

软件框图



主流程及其实现

在上次实验的基础上，增加下列内容。



主要功能模块及其实现

本次实验中要实现的功能模块分别按五个部分实现

内存初始化

- 判断起始内存地址
- 判断可用内存大小
- 写入可用内存信息



固定分区内存管理实现

- 建立内存结构体与块结构体
- 实现打印结构体
- 实现分块初始化
- 实现分配空闲块
- 实现释放块
- 实现调用
- 调试



Shell增加新命令功能

- 调用动态分配内存
- 实现新增命令功能



动态分区内存管理实现

- 建立内存结构体与块结构体
- 实现打印结构体
- 实现分块初始化
- 实现分配空闲块
- 实现释放块
- 实现调用
- 调试



管理头文件

- 核对框架已给头文件
- 添加颜色宏头文件

源代码说明

在本次实验的根目录下，除了配置了 makefile 与 DS 文件，还有 4 个文件夹：multibootheader、myOS、output、userApp。multibootheader 中放置了有关 multibootheader 协议的 S 文件。在 output 中，原本为空文件夹，用来输出我们编译后生成的文件。在该文件夹中的目录结构与根目录相似，主要是为了对每个文件输出时不产生混淆。在 userApp 中存放的是 main 文件、memTestCase 文件与该文件夹下的 makefile 文件，main 文件是用来测试本次实验的功能的入口，memTestCase 文件则是用来测试本次实验中实现的内存管理功能的。在 myOS 文件夹中存放了本次实验的主要源文件。直接存放在此该目录下的文件有 DS 文件、该文件夹下的 makefile 文件、链接器文件、一个配置地址的 S 文件和一个包含了 main 函数的与 multibootheader 中的 S 文件相关联的 C 文件。这个文件也是从汇编到 C 的转变。在该目录下则有 6 个实现相关功能的文件夹：dev、i386、printk、kernel、lib、include。每个文件夹下都有一个相应的 makefile 文件以及实现功能的 C 语言源文件。相比于上一次实验，框架中 include 文件夹下已经将之前所有的 myOS 中的头文件给包含在了其中，除此之外，我自行添加了颜色宏的头文件，方便用宏定义写入 myPrint 的颜色参数。lib 中的 String 文件更新了相关的一些字符串处理功能，vsprintf 里更新了一些例如十六进制数的格式化处理函数等。kernel 文件夹中增加 pMemInit、dPartition 和 eFPartition 代码文件以实现内存管理。每个子目录下的 makefile 文件的输出目录都是在 output 中的同名目录。下面展示相关源代码。

```
pMemInit.c* 中 X
杂项文件 - 无配置 (全局范围)
1  #include "../include/myPrintk.h"
2  #include "../include/mem.h"
3  #include "../include/io.h"
4
5  unsigned long pMemStart;           //可用的内存的起始地址
6  unsigned long pMemSize;           //可用的大小
7
8  void memTest(unsigned long start, unsigned long grainSize)
9  {
10     //检测范围
11     if (start <= 1 << 20)
12         start = 1 << 20;
13     if (grainSize < (1 << 10))
14         grainSize = 1 << 10;
15     unsigned long addr = start;
16     //每个步长检测
17     unsigned short a;
18     unsigned long i = 0;
19     for (; i++)
20     {
21         //检测头两字节
22         a = *((short int*)addr);
23         *((unsigned short*)addr) = (unsigned short)0xAA55;
24         if (*((unsigned short*)addr) != (unsigned short)0xAA55)
25         {
26             break;
27         }
28         *((unsigned short*)addr) = (unsigned short)0x55AA;
29         if (*((unsigned short*)addr) != (unsigned short)0x55AA)
30         {
31             break;
32         }
33         //写回
34         *((unsigned short*)addr) = a;
```

根据老师给的 PPT 中的检测内存算法实现检测内存功能，图 1

```

34      //检测后两字节
35      addr += grainSize - 2;
36      a = *((short int*)addr);
37      *((unsigned short*)addr) = (unsigned short)0xAA55;
38      if (*((unsigned short*)addr) != (unsigned short)0xAA55)
39          break;
40      *((unsigned short*)addr) = (unsigned short)0x55AA;
41      if (*((unsigned short*)addr) != (unsigned short)0x55AA)
42          break;
43      //写回
44      *((short int*)addr) = a;
45      addr += 2;
46  }
47  //打印信息
48  pMemStart = start;
49  pMemSize = addr - start;
50  myPrintk(GREY, "MemStart: 0x%x \n", pMemStart);
51  myPrintk(GREY, "MemSize: 0x%x \n", pMemSize);
52  }
53
54  extern unsigned long _end;
55  void pMemInit(void)
56  {
57      unsigned long _end_addr = (unsigned long)&_end;
58      memTest(0x100000, 0x1000);
59      myPrintk(GREY, "_end: 0x%x \n", _end_addr);
60      if (pMemStart <= _end_addr)
61      {
62          pMemSize -= _end_addr - pMemStart;
63          pMemStart = _end_addr;
64      }
65      pMemHandler = dPartitionInit(pMemStart, pMemSize);
66  }
67

```

根据老师给的 PPT 中的检测内存算法实现检测内存功能，图 2

```

eFPartition.c  dPartition.c  pMemInit.c
杂项文件 - 无配置  (全局范围)  showEEB(EEB * eeb)
1  #include "../include/myPrintk.h"
2  #define assign_byte (4) //对齐字节
3  #define eFPartitionsize (sizeof(eFPartition)) //内存结构体大小
4  #define EEBsize (sizeof(EEB)) //块结构体大小
5
6  // 一个EEB表示一个空闲可用的Block
7  typedef struct
8  {
9      unsigned long next_start;
10     int isfree;
11 } EEB;
12
13 void showEEB(EEB* eeb)
14 {
15     myPrintk(GREY, "EEB(address=0x%x, isfree=%d, next=0x%x)\n", eeb, eeb->isfree, eeb->next_start);
16 }
17
18 //eFPartition是表示整个内存的数据结构
19 typedef struct
20 {
21     unsigned long totalN;
22     unsigned long perSize; //unit: byte
23     unsigned long firstEEB;
24 } eFPartition;
25
26 //打印内存整体信息
27 void showeFPartition(eFPartition* efp)
28 {
29     myPrintk(PURPLE, "eFPartition(start=0x%x, totalN=0x%x, perSize=0x%x, firstEEB=0x%x)\n", efp, efp->totalN, efp->perSize, efp->firstEEB);
30 }
31

```

实现固定大小分区内存管理，建立结构体信息，主要思想是用 isfree 标记是否利用了内存。

```

32     //打印内存信息
33     void eFPartitionWalkByAddr(unsigned long efpHandler)
34     {
35         efpHandler -= eFPartitionsSize;
36         //打印内存整体信息
37         showeFPartition((eFPartition*)efpHandler);
38         //打印块信息
39         unsigned long p = ((eFPartition*)efpHandler)->firstEEB;
40         EEB* eebpoint;
41         while (p != 0)
42         {
43             eebpoint = (EEB*)p;
44             showEEB(eebpoint);
45             p = eebpoint->next_start;
46         }
47     }

```

实现打印固定大小分区内存的信息

```

49     //计算内存大小
50     unsigned long eFPartitionTotalSize(unsigned long perSize, unsigned long n)
51     {
52         return (((perSize - 1) / assign_byte) + 1) * assign_byte + EEBSIZE * n + eFPartitionsSize;
53     }
54
55     //内存初始化
56     unsigned long eFPartitionInit(unsigned long start, unsigned long perSize, unsigned long n)
57     {
58         myPrintk(DARK_GREEN, "The size of eFPartitionpoint is 0x%x bytes, the size of EEB is 0x%x bytes\n", eFPartitionsSize, EEBSIZE);
59         //创建一个eFPartition结构体
60         eFPartition* partitionpoint = (eFPartition*)start;
61         unsigned long realperSize = ((perSize - 1) / assign_byte + 1) * assign_byte + EEBSIZE;
62         partitionpoint->perSize = realperSize;
63         partitionpoint->totalN = n;
64         unsigned long p = start + eFPartitionsSize;
65         partitionpoint->firstEEB = p;
66         //对每一块的内存创建EEB连成链
67         for (int i = 0; i < n; i++)
68         {
69             EEB* eebpoint = (EEB*)p;
70             p += realperSize;
71             eebpoint->isfree = 1;
72             if (i < n - 1)
73                 eebpoint->next_start = p;
74             else
75                 eebpoint->next_start = 0;
76         }
77         return partitionpoint->firstEEB;
78     }

```

固定大小分区内存的初始化

```

80 //分配一个空闲块的内存并返回相应的地址
81 unsigned long eFPartitionAlloc(unsigned long EFPHandler)
82 {
83     EFPHandler -= eFPartitionsSize;
84     //记录返回地址
85     unsigned long p = ((eFPartition*)EFPHandler)->firstEEB;
86     EEB* eebpoint;
87     //检测下一个空闲块
88     while (p != 0)
89     {
90         eebpoint = (EEB*)p;
91         if (eebpoint->isfree == 1)
92         {
93             eebpoint->isfree = 0;
94             return p + EEBsize;
95         }
96         p = eebpoint->next_start;
97     }
98     return 0;
99 }
100
101 unsigned long eFPartitionFree(unsigned long EFPHandler, unsigned long mbStart)
102 {
103     EFPHandler -= eFPartitionsSize;
104     mbStart -= EEBsize;
105     //查找指定块
106     unsigned long p = ((eFPartition*)EFPHandler)->firstEEB;
107     EEB* eebpoint;
108     while (p != 0)
109     {
110         myPrintk(GREEN, "now addr is 0x%x\n", p);
111         eebpoint = (EEB*)p;
112         if (p == mbStart)
113         {
114             eebpoint->isfree = 1;
115             return 1;
116         }
117         p = eebpoint->next_start;
118     }
119     return 0;
120 }
121

```

实现固定大小分区内存管理的分配与释放。isfree 标志内存是否被占用以避免操作块链表这样的复杂结构。


```
dPartition.c  x  pMemInit.c
杂项文件 - 无配置  _unnamed_struct_0018_2

1  #include "../../include/myPrintk.h"
2  #define assign_byte (4) //对齐字节
3  #define dPartitionsize (sizeof(dPartition)) //内存结构体大小
4  #define EMBsize (sizeof(EMB)) //块结构体大小
5
6  //整个动态分区内存的数据结构
7  typedef struct
8  {
9      unsigned long size;
10     unsigned long EMBStart;
11 } dPartition;
12
13 void showdPartition(dPartition* dp)
14 {
15     myPrintk(PURPLE, "dPartition(start=0x%x, size=0x%x, EMBStart=0x%x)\n", dp, dp->size, dp->EMBStart);
16 }
17
18 //每一个block的数据结构
19 typedef struct
20 {
21     unsigned long size;
22     union
23     {
24         unsigned long nextStart;
25         unsigned long userData;
26     };
27     int isfree;
28 } EMB;
29
30 void showEMB(EMB* emb)
31 {
32     myPrintk(BLUE, "EMB(start=0x%x, size=0x%x, nextStart=0x%x, isfree=%d)\n", emb, emb->size, emb->nextStart, emb->isfree);
33 }
34
```

实现动态内存管理的结构体建立

```

35 //内存初始化
36 unsigned long dPartitionInit(unsigned long start, unsigned long totalSize)
37 {
38     //判断大小
39     if (totalSize <= dPartitionsize + EMBsize)
40         return 0;
41     //打印内存相关结构体的大小信息
42     myPrintk(DARK_GREEN, "The size of dPartitionpoint is 0x%x bytes, the size of EMB is 0x%x bytes\n", dPartitionsize, EMBsize);
43     //创建一个dPartition结构体
44     dPartition* dPartitionpoint = (dPartition*)start;
45     dPartitionpoint->EMBStart = start + dPartitionsize;
46     dPartitionpoint->size = totalSize - dPartitionsize;
47     //分配整块EMB
48     EMB* embpoint = (EMB*)dPartitionpoint->EMBStart;
49     embpoint->size = totalSize - dPartitionsize;
50     embpoint->nextStart = 0;
51     embpoint->isfree = 1;
52     return dPartitionpoint->EMBStart;
53 }
54
55 void dPartitionWalkByAddr(unsigned long dp)
56 {
57     dp -= dPartitionsize;
58     //打印内存整体信息
59     showdPartition((dPartition*)dp);
60     //遍历打印块信息
61     unsigned long p = ((dPartition*)dp)->EMBStart;
62     EMB* embpoint;
63     while (p != 0)
64     {
65         embpoint = (EMB*)p;
66         showEMB(embpoint);
67         p = embpoint->nextStart;
68     }
69 }

```

实现动态内存管理的内存信息打印与初始化

```

76  unsigned long dPartitionAllocFirstFit(unsigned long dp, unsigned long size)
77  {
78      dp -= dPartitionsSize;
79      //计算对齐所需大小
80      unsigned long realSize = ((size - 1) / asize_byte + 1) * asize_byte + EMSize;
81      //first fit查找块
82      unsigned long p = ((dPartition*)dp)->EMStart;
83      EMB* embpoint;
84      while (p != 0)
85      {
86          embpoint = (EMB*)p;
87          if (embpoint->isfree == 1 && embpoint->size >= realSize)
88          {
89              embpoint->isfree = 0;
90              //考虑是否分割块
91              if (embpoint->size > realSize + EMSize)
92              {
93                  ((EMB*)(p + realSize))->isfree = 1;
94                  ((EMB*)(p + realSize))->nextStart = embpoint->nextStart;
95                  ((EMB*)(p + realSize))->size = embpoint->size - realSize;
96                  embpoint->nextStart = p + realSize;
97                  embpoint->size = realSize;
98              }
99              return p + EMSize;
100          }
101          p = embpoint->nextStart;
102      }
103      return 0;
104  }
105

```

实现动态内存管理的分配功能。利用 isfree 与 size，通过第一适应来确定空闲块

```

106 //按照对应的fit的算法释放空间
107 unsigned long dPartitionFreeFirstFit(unsigned long dp, unsigned long
108 {
109     dp -= dPartitionsSize;
110     start -= EMBsize;
111     unsigned long p = ((dPartition*)dp)->EMBStart;
112     EMB* embpoint;
113     unsigned long lastEMBSize = 0;
114     while (p != 0)
115     {
116         embpoint = (EMB*)p;
117         //检测到地址
118         if (p == start)
119         {
120             //向上合并
121             if (lastEMBSize > 0)
122             {
123                 unsigned long lastp = p - lastEMBSize;
124                 EMB* lastembpoint = (EMB*)lastp;
125                 lastembpoint->size += embpoint->size;
126                 lastembpoint->nextStart = embpoint->nextStart;
127                 p = lastp;
128                 embpoint = (EMB*)lastp;
129             }
130             else
131                 embpoint->isfree = 1;
132             //向下合并
133             if (embpoint->nextStart != 0)
134             {
135                 unsigned long nextp = embpoint->nextStart;
136                 EMB* nextembpoint = (EMB*)nextp;
137                 if (nextembpoint->isfree == 1)
138                 {
139                     embpoint->size += nextembpoint->size;
140                     embpoint->nextStart = nextembpoint->nextStart;
141                 }
142             }
143             return 1;
144         }

```

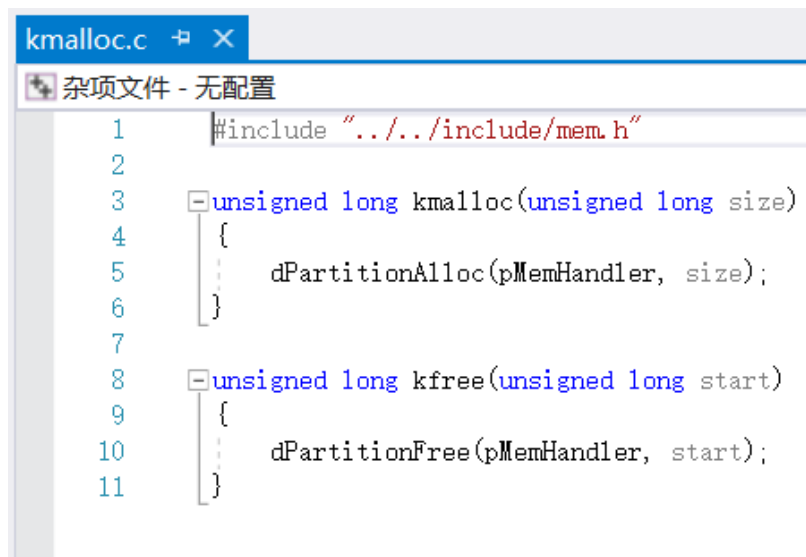
实现动态内存管理的释放功能，实现释放时相比固定大小分区的内存管理，还需要考虑合并空闲块，分别考虑向上与向下合并，其中向上合并时额外记录上一个块的大小。

```

145     //若当前块空闲则记录大小
146     if (embpoint->isfree == 1)
147         lastEMBsize = embpoint->size;
148     else
149         lastEMBsize = 0;
150     p = embpoint->nextStart;
151 }
152 return 0;
153 }
154
155 unsigned long dPartitionAlloc(unsigned long dp, unsigned long size)
156 {
157     return dPartitionAllocFirstFit(dp, size);
158 }
159
160 unsigned long dPartitionFree(unsigned long dp, unsigned long start)
161 {
162     return dPartitionFreeFirstFit(dp, start);
163 }
164

```

实现动态内存管理的分配功能，此处展示了记录当前内存块大小以便下一个内存块需要合并时使用。然后实现了动态内存管理的分配、释放调用。



The screenshot shows a code editor window titled 'kmalloc.c'. The editor contains the following code:

```

1  #include "../include/mem.h"
2
3  unsigned long kmalloc(unsigned long size)
4  {
5      dPartitionAlloc(pMemHandler, size);
6  }
7
8  unsigned long kfree(unsigned long start)
9  {
10     dPartitionFree(pMemHandler, start);
11 }

```

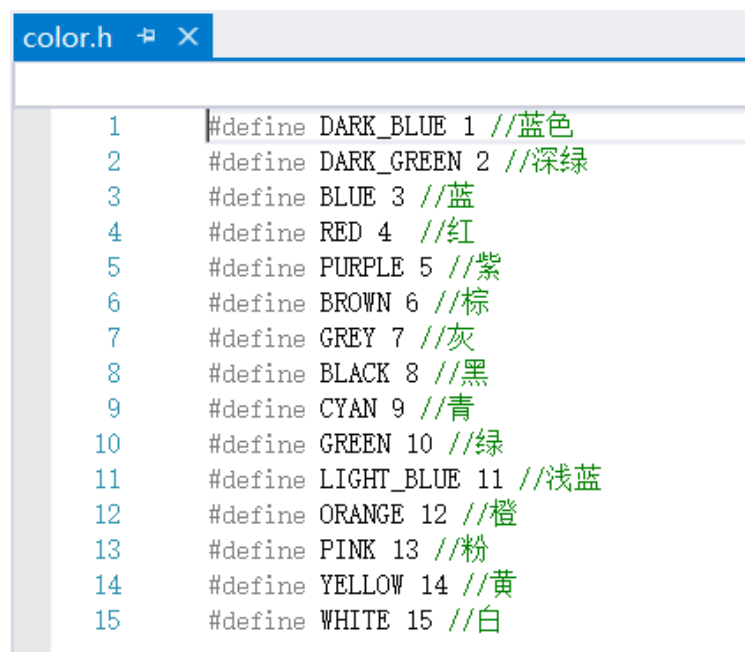
实现 kmalloc, malloc 的实现类似

```

53     void addNewCmd(unsigned char* cmd,
54                   int (*func)(int argc, unsigned char** argv),
55                   void (*help_func)(void),
56                   unsigned char* description)
57     {
58         Cmd* newcmd = (Cmd*)malloc(sizeof(Cmd));
59         strcpy(cmd, newcmd->cmd);
60         strcpy(description, newcmd->description);
61         newcmd->func = func;
62         newcmd->help_func = help_func;
63         newcmd->nextCmd = NULL;
64         //第一次创建
65         if (ourCmds == NULL)
66         {
67             ourCmds = newcmd;
68         }
69         else
70         {
71             //遍历
72             Cmd* cmdlist = ourCmds;
73             while (cmdlist->nextCmd != NULL)
74             {
75                 cmdlist = cmdlist->nextCmd;
76             }
77             cmdlist->nextCmd = newcmd;
78         }
79     }

```

实现 shell 动态添加命令。先动态申请内存，再进行赋值，然后添加到命令集的尾部

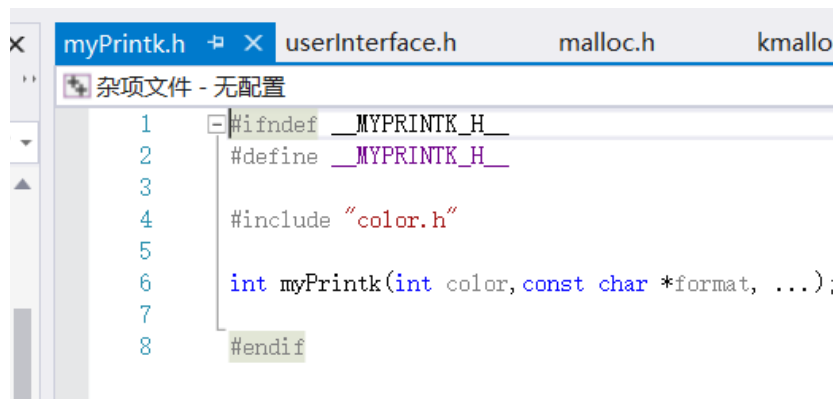


```

color.h  + x
1  #define DARK_BLUE 1 //蓝色
2  #define DARK_GREEN 2 //深绿
3  #define BLUE 3 //蓝
4  #define RED 4 //红
5  #define PURPLE 5 //紫
6  #define BROWN 6 //棕
7  #define GREY 7 //灰
8  #define BLACK 8 //黑
9  #define CYAN 9 //青
10 #define GREEN 10 //绿
11 #define LIGHT_BLUE 11 //浅蓝
12 #define ORANGE 12 //橙
13 #define PINK 13 //粉
14 #define YELLOW 14 //黄
15 #define WHITE 15 //白

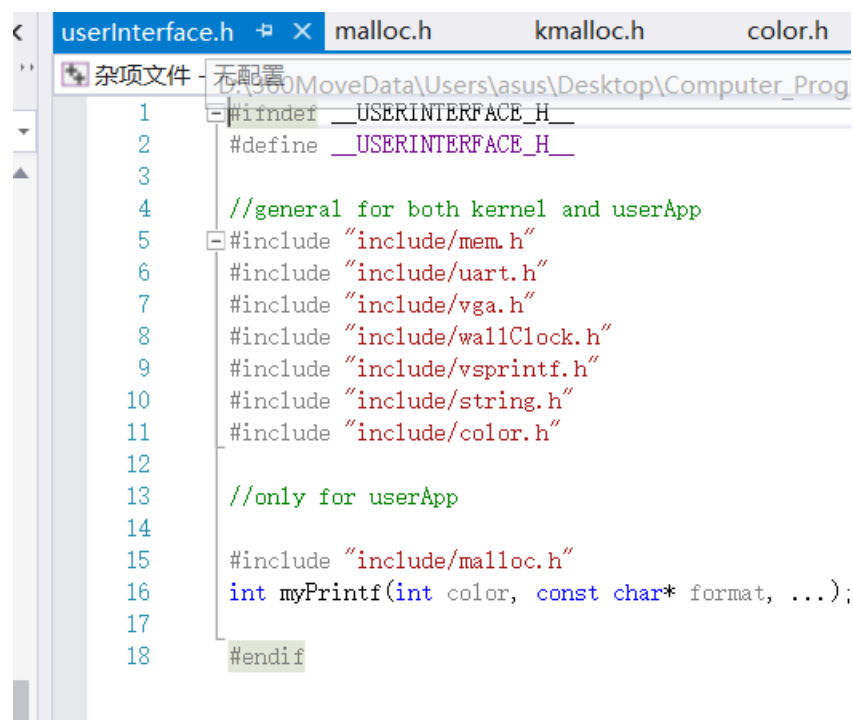
```

颜色宏头文件，对应了 myprint 中的颜色参数



```
1 #ifndef _MYPRINTK_H_
2 #define _MYPRINTK_H_
3
4 #include "color.h"
5
6 int myPrintk(int color, const char *format, ...);
7
8 #endif
```

在内核态的打印函数的头文件中添加了颜色宏头文件，这样在调用 printk 时可以直接输入颜色宏代替数字参数。



```
1 #ifndef _USERINTERFACE_H_
2 #define _USERINTERFACE_H_
3
4 //general for both kernel and userApp
5 #include "include/mem.h"
6 #include "include/uart.h"
7 #include "include/vga.h"
8 #include "include/wallClock.h"
9 #include "include/vsprintf.h"
10 #include "include/string.h"
11 #include "include/color.h"
12
13 //only for userApp
14
15 #include "include/malloc.h"
16 int myPrintf(int color, const char* format, ...);
17
18 #endif
```

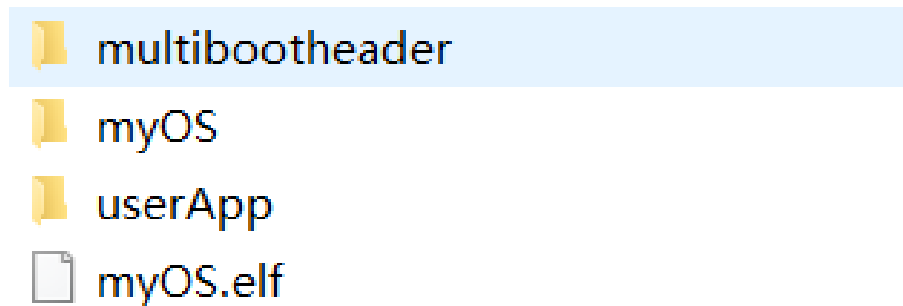
用户态的头文件被整合。用户态的打印函数的头文件中调用颜色宏头文件，使得 printf 也可以使用宏定义的颜色参数

代码布局说明

所有的引导模块将按页（4KB）边界对齐，物理内存地址从 1M 处开始。本次实验的内存管理统一以 4 字节对齐，不足 4 字节补全。

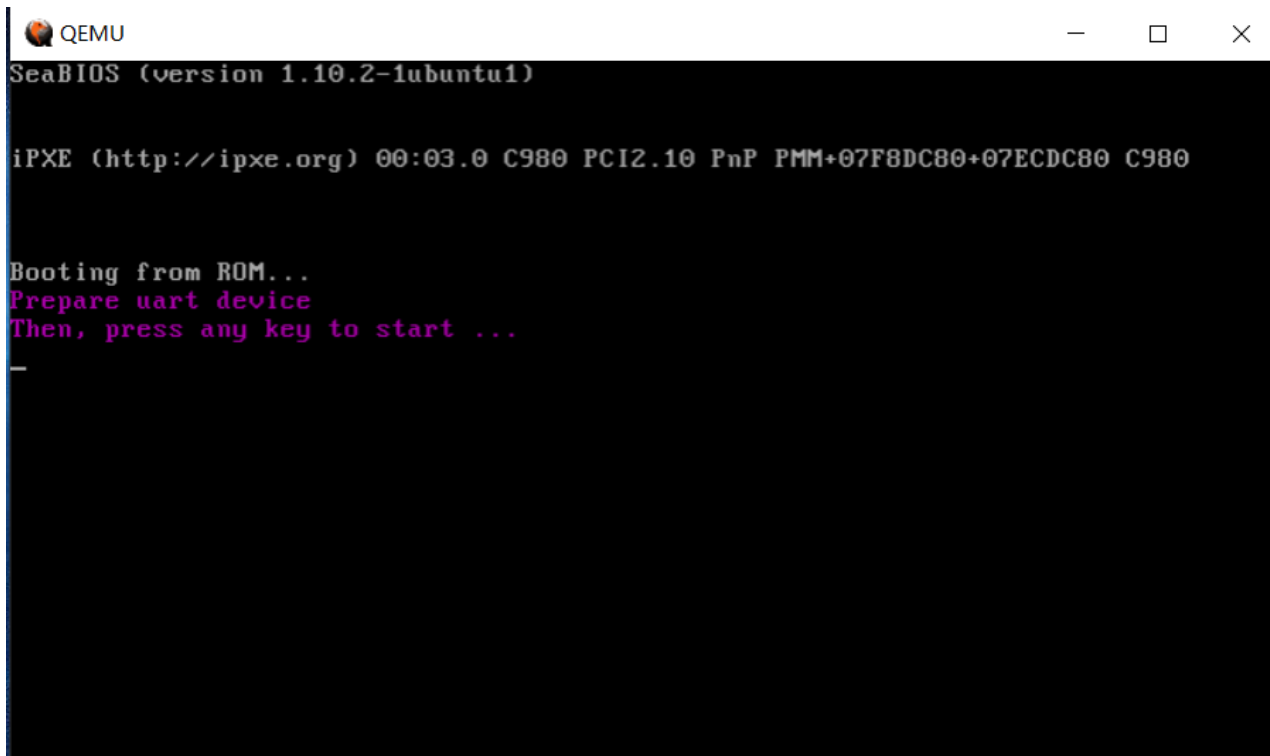
编译过程说明：

在 Ubuntu 中先搜索到 lab4 的目录，然后通过指令 make 完成编译，可以看到在 output 目录中的对应目录中分别输出了与根目录下对应文件相同文件。



运行和运行结果说明：

在 Ubuntu 中通过 QEMU 启动已经编译生成的 bin 文件，得到 Linux 的图形化界面运行结果，然后再通过 Ubuntu 启动一个交互界面，用于输入与输出。

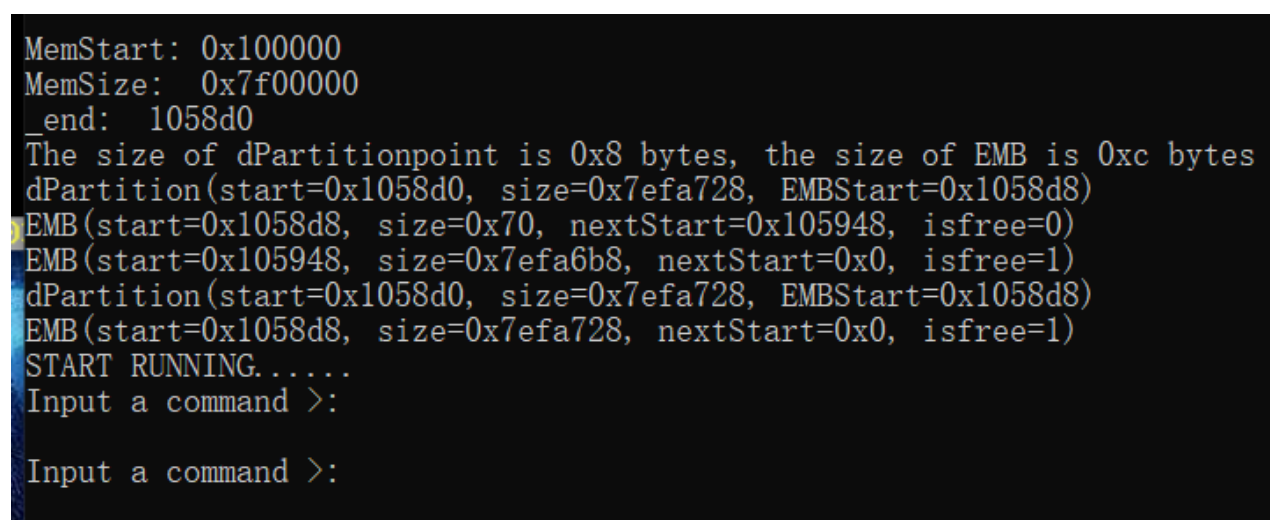
A screenshot of a QEMU window showing the SeaBIOS boot screen. The window title is 'QEMU'. The text on the screen includes 'SeaBIOS (version 1.10.2-1ubuntu1)', 'iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DC80+07ECDC80 C980', 'Booting from ROM...', 'Prepare uart device', and 'Then, press any key to start ...'.

```
QEMU
SeaBIOS (version 1.10.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DC80+07ECDC80 C980

Booting from ROM...
Prepare uart device
Then, press any key to start ...
```

刚运行内核时的 QEMU 界面。本次实验中在进入 main 前加入了一个验证输入任意键的过程。

A screenshot of a QEMU screen showing memory initialization details. The text includes 'MemStart: 0x100000', 'MemSize: 0x7f00000', '_end: 1058d0', 'The size of dPartitionpoint is 0x8 bytes, the size of EMB is 0xc bytes', 'dPartition(start=0x1058d0, size=0x7efa728, EMBStart=0x1058d8)', 'EMB(start=0x1058d8, size=0x70, nextStart=0x105948, isfree=0)', 'EMB(start=0x105948, size=0x7efa6b8, nextStart=0x0, isfree=1)', 'dPartition(start=0x1058d0, size=0x7efa728, EMBStart=0x1058d8)', 'EMB(start=0x1058d8, size=0x7efa728, nextStart=0x0, isfree=1)', 'START RUNNING.....', 'Input a command >:', and 'Input a command >:'.

```
MemStart: 0x100000
MemSize: 0x7f00000
_end: 1058d0
The size of dPartitionpoint is 0x8 bytes, the size of EMB is 0xc bytes
dPartition(start=0x1058d0, size=0x7efa728, EMBStart=0x1058d8)
EMB(start=0x1058d8, size=0x70, nextStart=0x105948, isfree=0)
EMB(start=0x105948, size=0x7efa6b8, nextStart=0x0, isfree=1)
dPartition(start=0x1058d0, size=0x7efa728, EMBStart=0x1058d8)
EMB(start=0x1058d8, size=0x7efa728, nextStart=0x0, isfree=1)
START RUNNING.....
Input a command >:
Input a command >:
```

启动了 screen 界面后按下任意键的结果，可以看到内存初始化同时利用动态内存申请了一块内存并释放前后的内存信息，EMB 块从两块变成一块，以及其起始地址可占用大小。

```

Input a command >:cmd
cmd
list all registered commands:
command name: description
      cmd: list all registered commands
      help: help [cmd]
testMalloc1: Malloc, write and read.
testMalloc2: Malloc, write and read.
testMalloc3: Kmalloc, write and read.
maxMallocSizeNow: MAX_MALLOC_SIZE always changes. What's the value Now?
      testDP1: Init a dPatition(size=0x100). [Alloc,Free]* with step = 0x20
      testDP2: Init a dPatition(size=0x100). A:B:C:- ==> -:B:C:- ==> -:C:- ==> - .
      testDP3: Init a dPatition(size=0x100). A:B:C:- ==> A:B:- ==> A:- ==> - .
      testeFP: Init a eFPatition. Alloc all and Free all.
Input a command >:

```

输出 cmd 查看本次实验中动态添加的所有命令

```

      testeFP: Init a eFPatition. Alloc all and Free all.
Input a command >:testMalloc1
testMalloc1
We allocated 2 buffers.
BUF1(size=19, addr=0x105ed4) filled with 17(*): *****
BUF2(size=24, addr=0x105ef4) filled with 22(#): #####

Input a command >:testMalloc2
testMalloc2
We allocated 2 buffers.
BUF1(size=9, addr=0x105ed4) filled with 9(+): ++++++++
BUF2(size=19, addr=0x105eec) filled with 19(,): ,, ,, ,, ,, ,, ,, ,, ,, ,,

Input a command >:tast
tast
UNKOWN command: tast
Input a command >:testMalloc3
testMalloc3
We kallocated 2 buffers.
BUF1(size=19, addr=0x105ed4) filled with 17(*): *****
BUF2(size=24, addr=0x105ef4) filled with 22(#): #####

Input a command >:

```

对前 3 个内存测试指令进行调用的结果，其中前两个是 malloc，第三个是 kmalloc

```

Input a command >:maxMallocSizeNow
maxMallocSizeNow
MAX_MALLOC_SIZE: 0x7efb000 (with step = 0x1000);
Input a command >:testdP1
testdP1
We had successfully malloc() a small memBlock (size=0x100, addr=0x105ed4)
It is initialized as a very small dPartition;
The size of dPartitionpoint is 0x8 bytes, the size of EMB is 0xc bytes
dPartition(start=0x105ecc, size=0x105fd4, EMBStart=0x0)
Alloc a memBlock with size 0x10, success(addr=0x105ee8)!.....Relaesed;
Alloc a memBlock with size 0x20, success(addr=0x105ee8)!.....Relaesed;
Alloc a memBlock with size 0x40, success(addr=0x105ee8)!.....Relaesed;
Alloc a memBlock with size 0x80, success(addr=0x105ee8)!.....Relaesed;
Alloc a memBlock with size 0x100, failed!
Now, converse the sequence.
Alloc a memBlock with size 0x100, failed!
Alloc a memBlock with size 0x80, success(addr=0x105ee8)!.....Relaesed;
Alloc a memBlock with size 0x40, success(addr=0x105ee8)!.....Relaesed;
Alloc a memBlock with size 0x20, success(addr=0x105ee8)!.....Relaesed;
Alloc a memBlock with size 0x10, success(addr=0x105ee8)!.....Relaesed;

```

测试最大可分配内存以及第一个固定大小内存分配

```

Input a command >:testdP2
testdP2
We had successfully malloc() a small memBlock (size=0x100, addr=0x105ed4);
It is initialized as a very small dPartition;
The size of dPartitionpoint is 0x8 bytes, the size of EMB is 0xc bytes
dPartition(start=0x105ecc, size=0x105fd4, EMBStart=0x0)
Now, A:B:C:- ==> -:B:C:- ==> -:C- ==> - .
Alloc memBlock A with size 0x10: success(addr=0x105ee8)!
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0x1c, nextStart=0x105ef8, isfree=0)
EMB(start=0x105ef8, size=0xdc, nextStart=0x0, isfree=1)
Alloc memBlock B with size 0x20: success(addr=0x105f04)!
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0x1c, nextStart=0x105ef8, isfree=0)
EMB(start=0x105ef8, size=0x2c, nextStart=0x105f24, isfree=0)
EMB(start=0x105f24, size=0xb0, nextStart=0x0, isfree=1)
Alloc memBlock C with size 0x30: success(addr=0x105f30)!
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0x1c, nextStart=0x105ef8, isfree=0)
EMB(start=0x105ef8, size=0x2c, nextStart=0x105f24, isfree=0)
EMB(start=0x105f24, size=0x3c, nextStart=0x105f60, isfree=0)
EMB(start=0x105f60, size=0x74, nextStart=0x0, isfree=1)
Now, release A.
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0x1c, nextStart=0x105ef8, isfree=1)
EMB(start=0x105ef8, size=0x2c, nextStart=0x105f24, isfree=0)
EMB(start=0x105f24, size=0x3c, nextStart=0x105f60, isfree=0)
EMB(start=0x105f60, size=0x74, nextStart=0x0, isfree=1)
Now, release B.
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0x48, nextStart=0x105f24, isfree=1)
EMB(start=0x105f24, size=0x3c, nextStart=0x105f60, isfree=0)
EMB(start=0x105f60, size=0x74, nextStart=0x0, isfree=1)
At last, release C.
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0xf8, nextStart=0x0, isfree=1)
Input a command >:

```

测试一个动态内存分配

```

testdP3
We had successfully malloc() a small memBlock (size=0x100, addr=0x105ed4);
It is initialized as a very small dPartition;
The size of dPartitionpoint is 0x8 bytes, the size of EMB is 0xc bytes
dPartition(start=0x105ecc, size=0x105fd4, EMBStart=0x0)
Now, A:B:C:- ==> A:B:- ==> A:- ==> - .
Alloc memBlock A with size 0x10: success(addr=0x105ee8)!
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0x1c, nextStart=0x105ef8, isfree=0)
EMB(start=0x105ef8, size=0xdc, nextStart=0x0, isfree=1)
Alloc memBlock B with size 0x20: success(addr=0x105f04)!
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0x1c, nextStart=0x105ef8, isfree=0)
EMB(start=0x105ef8, size=0x2c, nextStart=0x105f24, isfree=0)
EMB(start=0x105f24, size=0xb0, nextStart=0x0, isfree=1)
Alloc memBlock C with size 0x30: success(addr=0x105f30)!
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0x1c, nextStart=0x105ef8, isfree=0)
EMB(start=0x105ef8, size=0x2c, nextStart=0x105f24, isfree=0)
EMB(start=0x105f24, size=0x3c, nextStart=0x105f60, isfree=0)
EMB(start=0x105f60, size=0x74, nextStart=0x0, isfree=1)
Now, release C.
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0x1c, nextStart=0x105ef8, isfree=0)
EMB(start=0x105ef8, size=0x2c, nextStart=0x105f24, isfree=0)
EMB(start=0x105f24, size=0xb0, nextStart=0x0, isfree=1)
Now, release B.
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0x1c, nextStart=0x105ef8, isfree=0)
EMB(start=0x105ef8, size=0xdc, nextStart=0x0, isfree=1)
At last, release A.
dPartition(start=0x105ed4, size=0xf8, EMBStart=0x105edc)
EMB(start=0x105edc, size=0xf8, nextStart=0x0, isfree=1)
Input a command >:

```

测试另一个顺序释放的动态内存分配

```
EEB(address=0x105f30, isfree=0, next=0x105f58)
EEB(address=0x105f58, isfree=1, next=0x0)
Alloc memBlock D, start = 0x105f60: 0xdddddddd
eFPartition(start=0x105ed4, totalN=0x4, perSize=0x28, firstEEB=0x105ee0)
EEB(address=0x105ee0, isfree=0, next=0x105f08)
EEB(address=0x105f08, isfree=0, next=0x105f30)
EEB(address=0x105f30, isfree=0, next=0x105f58)
EEB(address=0x105f58, isfree=0, next=0x0)
Alloc memBlock E, failed!
eFPartition(start=0x105ed4, totalN=0x4, perSize=0x28, firstEEB=0x105ee0)
EEB(address=0x105ee0, isfree=0, next=0x105f08)
EEB(address=0x105f08, isfree=0, next=0x105f30)
EEB(address=0x105f30, isfree=0, next=0x105f58)
EEB(address=0x105f58, isfree=0, next=0x0)
Now, release A.
now addr is 0x105ee0
eFPartition(start=0x105ed4, totalN=0x4, perSize=0x28, firstEEB=0x105ee0)
EEB(address=0x105ee0, isfree=1, next=0x105f08)
EEB(address=0x105f08, isfree=0, next=0x105f30)
EEB(address=0x105f30, isfree=0, next=0x105f58)
EEB(address=0x105f58, isfree=0, next=0x0)
Now, release B.
now addr is 0x105ee0
now addr is 0x105f08
eFPartition(start=0x105ed4, totalN=0x4, perSize=0x28, firstEEB=0x105ee0)
EEB(address=0x105ee0, isfree=1, next=0x105f08)
EEB(address=0x105f08, isfree=1, next=0x105f30)
EEB(address=0x105f30, isfree=0, next=0x105f58)
EEB(address=0x105f58, isfree=0, next=0x0)
Now, release C.
now addr is 0x105ee0
now addr is 0x105f08
now addr is 0x105f30
eFPartition(start=0x105ed4, totalN=0x4, perSize=0x28, firstEEB=0x105ee0)
EEB(address=0x105ee0, isfree=1, next=0x105f08)
EEB(address=0x105f08, isfree=1, next=0x105f30)
EEB(address=0x105f30, isfree=1, next=0x105f58)
EEB(address=0x105f58, isfree=0, next=0x0)
Now, release D.
now addr is 0x105ee0
now addr is 0x105f08
now addr is 0x105f30
now addr is 0x105f58
eFPartition(start=0x105ed4, totalN=0x4, perSize=0x28, firstEEB=0x105ee0)
EEB(address=0x105ee0, isfree=1, next=0x105f08)
EEB(address=0x105f08, isfree=1, next=0x105f30)
EEB(address=0x105f30, isfree=1, next=0x105f58)
EEB(address=0x105f58, isfree=1, next=0x0)
Input a command >:
```

测试一个固定大小分区的内存分配与释放

遇到的问题解决方案：

1. 不能理解两种内存管理机制

通过网络工具查询和了解。

2. 不理解覆盖写入的操作方式

咨询助教并选择了直接用 C 语言的指针进行修改值。