

中国科学技术大学计算机学院
《操作系统原理与设计》实验报告



实验题目：lab4_Memory Management

学生姓名：胡毅翔

学生学号：PB18000290

完成日期：2020 年 5 月 15 日

计算机实验教学中心制

2019 年 09 月

实验目的

1. 实现内存检测，确定动态内存的范围。
2. 实现内存的动态分区管理机制和等大小分区管理机制。
3. 提供 `kmalloc/kfree` 及 `malloc/free` 两套接口，供内核和用户使用。
4. 提供 `addNewCmd()` 函数，用来增加新的命令行指令。

实验环境

1. PC 一台
2. Windows 系统
3. Ubuntu
4. QEMU
5. Xserver

软件框图

本实验的软件框图如图所示。软件层次分为 `multiboot_header`、`myOS` 和 `userApp` 三部分。

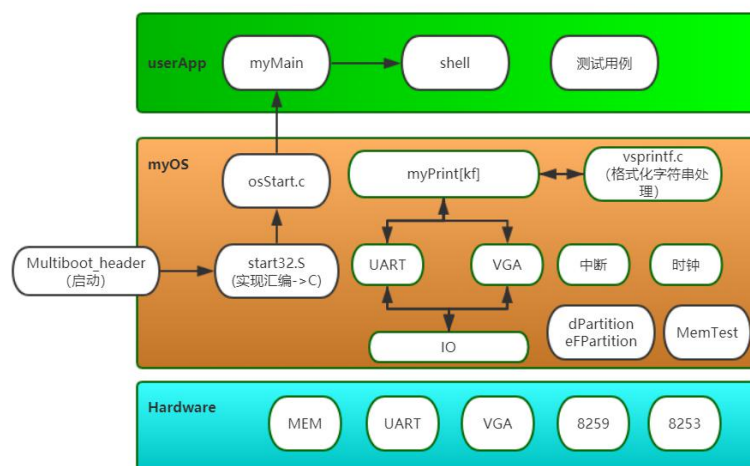


图 1

主流程



图 2

本实验的主流程如上图所示：

1. 在 `multiboot_header` 中完成系统的启动。
2. 在 `start32.S` 中准备好上下文，最后调用 `osStart.c` 把进入 `c` 程序。
3. 在 `osStart.c` 中完成初始化 8259A，初始化 8253，清屏及内存初始化等操作，调用 `myMain`，进入 `userApp` 部分。
4. 运行 `myMain` 中的代码，进行时钟设置，`shell` 初始化，内存测试初始化等操作，启动 `shell`。

5. 进入 shell 程序，等待命令的输入。

主要功能模块及其实现

内存检查的实现

该功能模块用于内存初始化时，判断可用空间的大小，主要目的是从 `start` 开始，按 `grainSize` 的步长，逐块读写，校对；若出错，说明已达到上界，退出循环，返回可用大小。

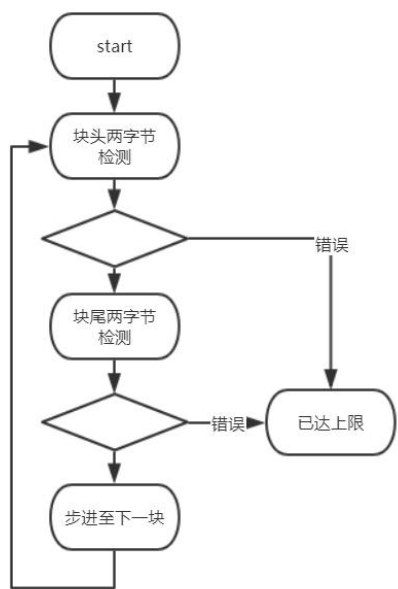


图 3

动态分区管理机制

该功能模块用于对检测出的动态内存块进行管理，主要目的是实现 `dPartitionInit()` 以及 `dPartitionAlloc()`, `dPartitionFree()` 函数。

`dPartitionInit()` 实现动态分区的初始化，将待分配的内存块，作为一块进行管理。

`dPartitionAlloc()` 实现分配，对相关的内存块大小，及表示下一空闲块的指针进行调整。

`dPartitionFree()` 实现释放，将指定内存进行释放，并对管理的数据结构进行调整。

`dPartitionAlloc()` 及 `dPartitionFree()` 经过包装后，即得到 `kmalloc()`, `kfree()`, `malloc()` 及 `free()` 函数。

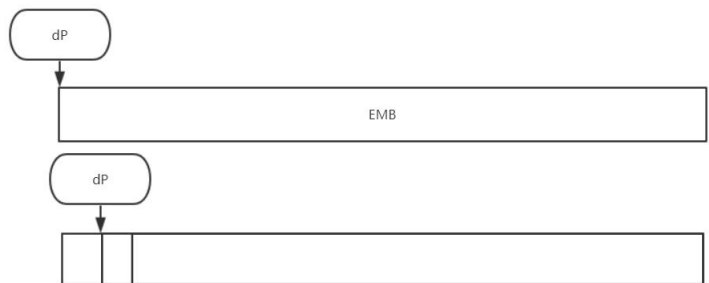


图 4

等大小分区管理机制

该功能模块用于对检测出的动态内存块进行等大小管理，主要目的是实现 `eFPartitionTotalSize()`, `eFPartitionInit()`, `eFPartitionAlloc()` 以及 `eFPartitionFree()` 函数。

eFPartitionTotalSize()实现对传入的大小的对齐，返回实际需要的大小。

eFPartitionInit()实现将所有块按顺序连接成链，便于分配和回收。

eFPartitionAlloc()实现块的分配，对链表进行重新链接，返回供使用的地址。

eFPartitionFree()实现块的逐一回收并重新链接至链表上。

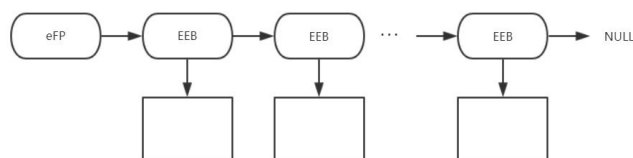


图 5

addNewCmd 的实现

该模块的主要功能是实现 shell 中的命令的动态添加。

实现过程为提供数据结构中的 `nextcmd` 链接新的指令,若为之后一个指令则该值设为 0; 添加命令时,将最后一个 `nextcmd` 赋值为新命令的地址,并将新命令的 `nextcmd` 域设为 0 即可。

源代码说明

目录组织

目录组织如下图所示，主目录下

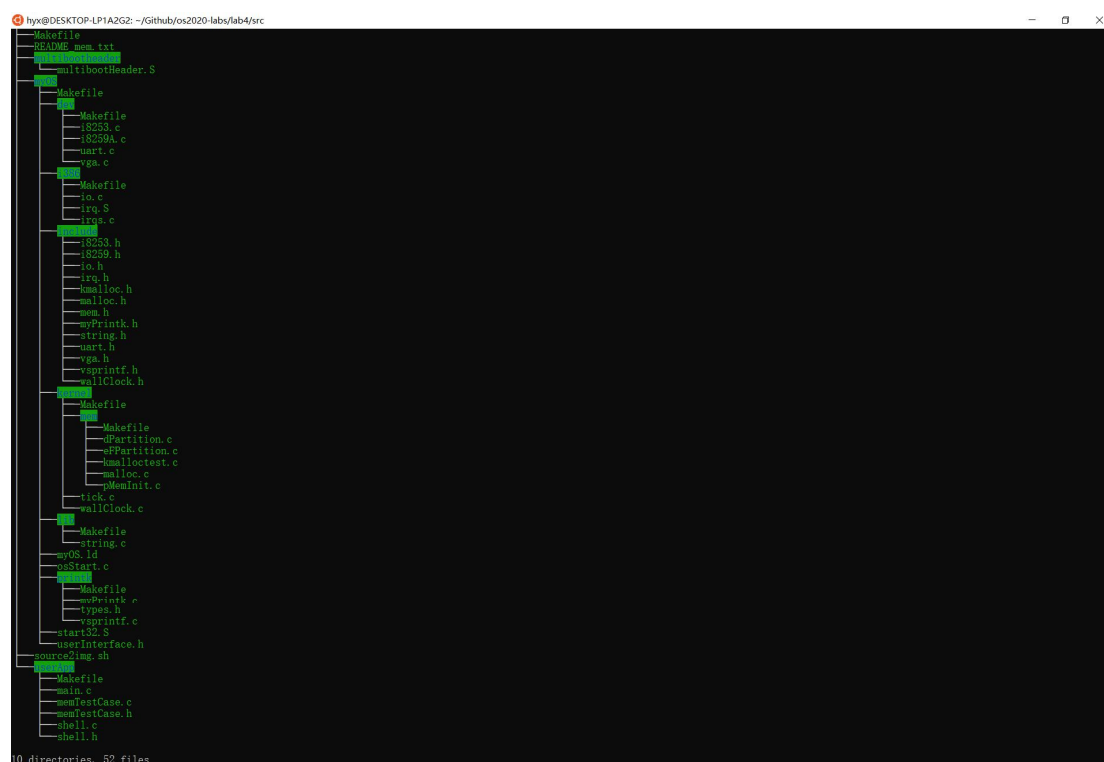


图 6

Makefile 组织

Makefile 组织结构如下:

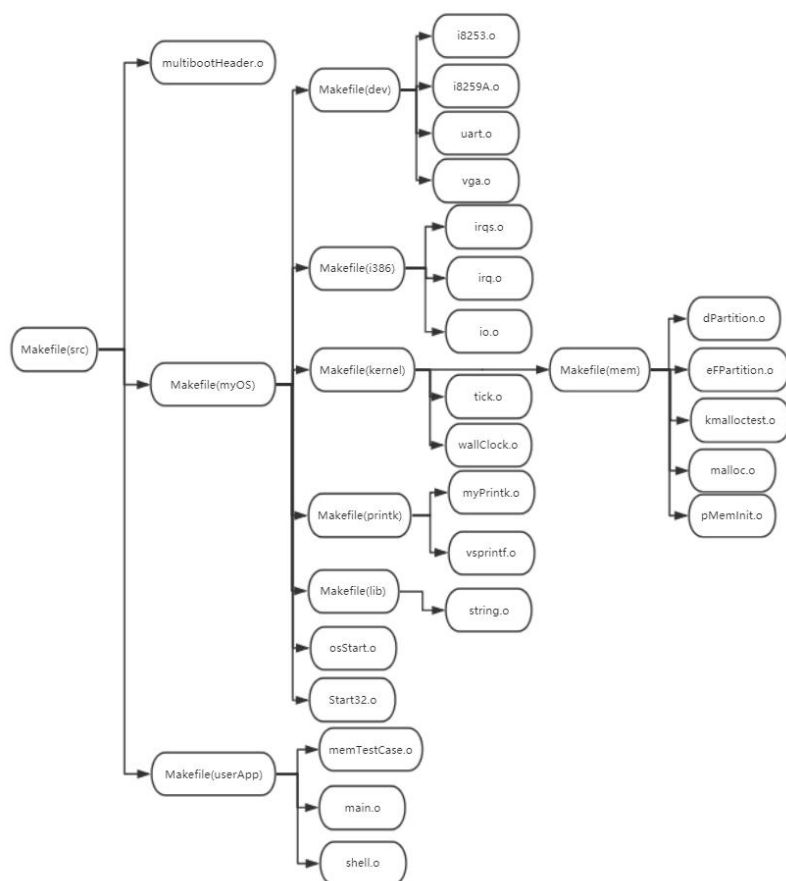


图 7

代码布局说明（地址空间）

从物理内存 1M 的位置开始放代码和数据，前面 12 个字节为 `multiboot_header`，向后对齐 8 个字节，放代码。再向后对齐 16 个字节，用于放初始化的数据（数据段）。在数据段之后，再向后对齐 16 个字节。之后为 BSS（Block Started by Symbol）段，用于存放程序中未初始化的全局变量和静态变量。并在 BSS 段后，再向后对齐 16 个字节。剩余部分为堆栈段，即为本次实验使用的动态内存空间。

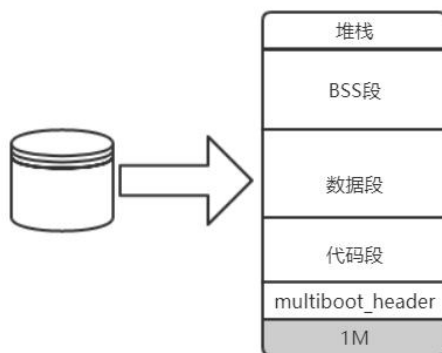


图 8

编译过程说明

```
ASM_FLAGS= -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector
```

- m32 用 32 位机器的编译器来编译这个文件
- pipe 使用管道代替编译中临时文档
- Wall 打开警告选项
- fasm 识别 asm 关键字
- g 使用调试器 GDB
- O1 优化生成代码
- fno-stack-protector 停止使用 stack-protector 功能

```
C_FLAGS = -m32 -fno-stack-protector -fno-pic -fno-builtin -g
```

- fpic 如果支持这种目标机,编译器就生成位置无关目标码,适用于共享库(shared library)

生成 multiHeader.o、osStart.o、start32.o、uart.o、vga.o、io.o、myPrintk.o、vsprintf.o 及 main.o 多个目标文件,链接生成 myOS.elf 文件。

运行和运行结果说明

输入 ./source2run.sh 指令后,编译,链接,生成 myOS.elf 文件并运行之,输入 sudo screen /dev/pts/0,通过 Ubuntu 输入。在 shell 程序中,执行测试命令。运行结果如下图:

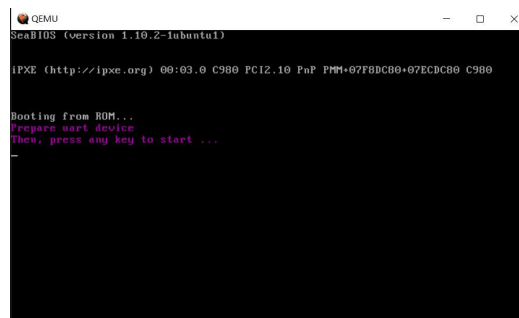


图 9

输入任意键启动。

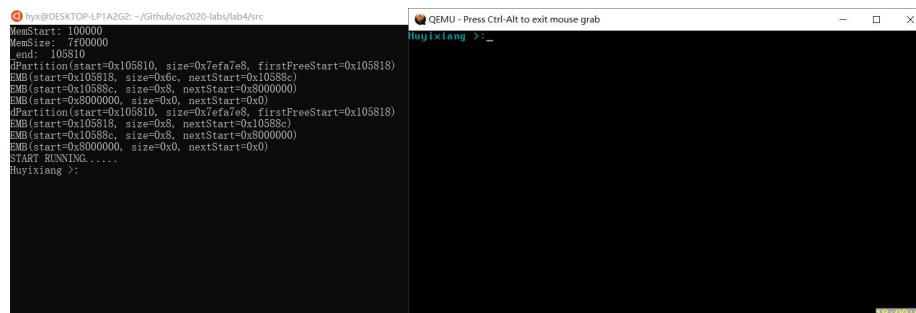


图 10

输入 cmd, 列出所有命令。

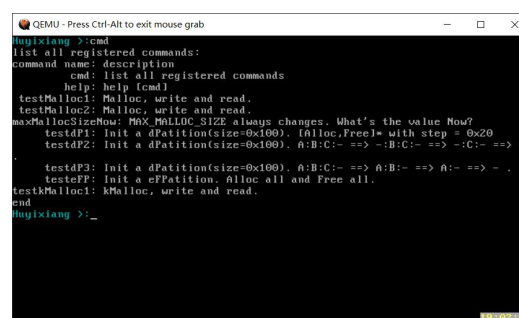
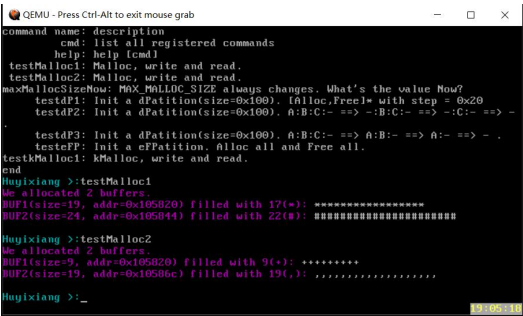


图 11

输入 testMalloc1, testMalloc2, 进行测试。



```
QEMU - Press Ctrl-Alt to exit mouse grab
Command name: description
cmd: list all registered commands
help: help [cmd]
testMalloc1: Malloc, write and read.
testMalloc2: Malloc, write and read.
maxMallocSizeNow: MAX_MALLOC_SIZE always changes. What's the value Now?
testdP1: Init a dPartition(size=0x100). (alloc,free) with step = 0x20
testdP2: Init a dPartition(size=0x100). A:B:C:- => -:B:C:- => -:C:- => -
testdP3: Init a dPartition(size=0x100). A:B:C:- => A:B:- => A:- => -
testeP: Init a ePartition. Alloc all and Free all.
testkMalloc1: kMalloc, write and read.
end
Huyixiang > testMalloc1
We allocated 2 buffers.
BUF1(size=19, addr=0x105820) filled with 17(*): *****
BUF2(size=24, addr=0x105844) filled with 22(#): *****
Huyixiang > testMalloc2
We allocated 2 buffers.
BUF1(size=9, addr=0x105820) filled with 9(+): *****
BUF2(size=19, addr=0x10580c) filled with 19(,): *****
Huyixiang > _
```

图 12

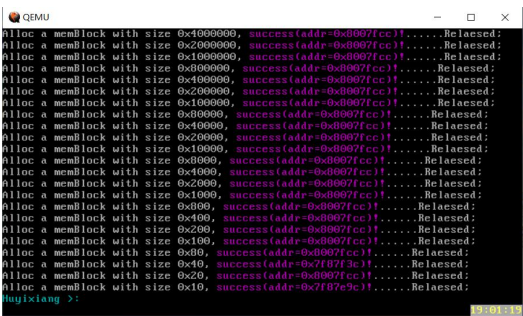
输入 maxMallocSizeNow, 进行测试。



```
QEMU
Huyixiang > maxMallocSizeNow
MAX_MALLOC_SIZE: 0x7efb000 (with step = 0x1000):
Huyixiang > _
```

图 13

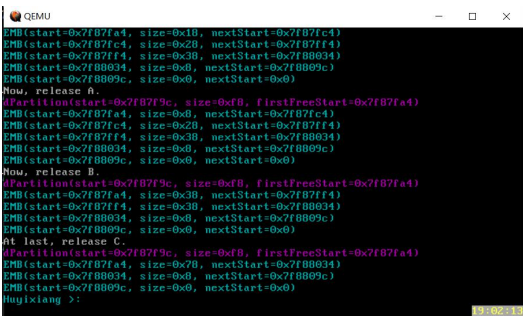
输入 testdP1, 进行测试。



```
QEMU
Alloc a memBlock with size 0x4000000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x2000000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x1000000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x800000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x400000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x200000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x100000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x80000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x40000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x20000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x10000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x8000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x4000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x2000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x1000, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x800, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x400, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x200, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x100, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x80, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x40, success(addr=0x7f073e)!.....Released:
Alloc a memBlock with size 0x20, success(addr=0x0007fcc)!.....Released:
Alloc a memBlock with size 0x10, success(addr=0x7f073e)!.....Released:
Huyixiang > _
```

图 14

输入 testdP2, 进行测试。



```
QEMU
EMB(start=0x7f07fa4, size=0x10, nextStart=0x7f07fc4)
EMB(start=0x7f07fc4, size=0x20, nextStart=0x7f07ff4)
EMB(start=0x7f07ff4, size=0x30, nextStart=0x7f08034)
EMB(start=0x7f08034, size=0x8, nextStart=0x7f0809c)
EMB(start=0x7f0809c, size=0x0, nextStart=0x0)
Now, release A.
dPartition(start=0x7f07fc, size=0xf0, firstFreeStart=0x7f07fa4)
EMB(start=0x7f07fa4, size=0x0, nextStart=0x7f07fc4)
EMB(start=0x7f07fc4, size=0x20, nextStart=0x7f07ff4)
EMB(start=0x7f07ff4, size=0x30, nextStart=0x7f08034)
EMB(start=0x7f08034, size=0x8, nextStart=0x7f0809c)
EMB(start=0x7f0809c, size=0x0, nextStart=0x0)
Now, release B.
dPartition(start=0x7f07fc, size=0xf0, firstFreeStart=0x7f07fa4)
EMB(start=0x7f07fa4, size=0x30, nextStart=0x7f07ff4)
EMB(start=0x7f07ff4, size=0x30, nextStart=0x7f08034)
EMB(start=0x7f08034, size=0x8, nextStart=0x7f0809c)
EMB(start=0x7f0809c, size=0x0, nextStart=0x0)
At last, release C.
dPartition(start=0x7f07fc, size=0xf0, firstFreeStart=0x7f07fa4)
EMB(start=0x7f07fa4, size=0x0, nextStart=0x7f08034)
EMB(start=0x7f08034, size=0x8, nextStart=0x7f0809c)
EMB(start=0x7f0809c, size=0x0, nextStart=0x0)
Huyixiang > _
```

图 15

输入 testdP3, 进行测试。

```
QEMU
EBD(start=0x7f87fa4, size=0x10, nextStart=0x7f87fc4)
EBD(start=0x7f87fc4, size=0x20, nextStart=0x7f87ff4)
EBD(start=0x7f87ff4, size=0x30, nextStart=0x7f88034)
EBD(start=0x7f88034, size=0x0, nextStart=0x7f8809c)
EBD(start=0x7f8809c, size=0x0, nextStart=0x0)
Now, release A.
dPartition(start=0x7f87f9c, size=0xf0, firstFreeStart=0x7f87fa4)
EBD(start=0x7f87fa4, size=0x0, nextStart=0x7f87fc4)
EBD(start=0x7f87fc4, size=0x20, nextStart=0x7f87ff4)
EBD(start=0x7f87ff4, size=0x30, nextStart=0x7f88034)
EBD(start=0x7f88034, size=0x0, nextStart=0x7f8809c)
EBD(start=0x7f8809c, size=0x0, nextStart=0x0)
Now, release B.
dPartition(start=0x7f87f9c, size=0xf0, firstFreeStart=0x7f87fa4)
EBD(start=0x7f87fa4, size=0x0, nextStart=0x7f87fc4)
EBD(start=0x7f87fc4, size=0x20, nextStart=0x7f87ff4)
EBD(start=0x7f87ff4, size=0x30, nextStart=0x7f88034)
EBD(start=0x7f88034, size=0x0, nextStart=0x7f8809c)
EBD(start=0x7f8809c, size=0x0, nextStart=0x0)
At last, release C.
dPartition(start=0x7f87f9c, size=0xf0, firstFreeStart=0x7f87fa4)
EBD(start=0x7f87fa4, size=0x70, nextStart=0x7f88034)
EBD(start=0x7f88034, size=0x0, nextStart=0x7f8809c)
EBD(start=0x7f8809c, size=0x0, nextStart=0x0)
huyixiang %
```

图 16

输入 testeFP，进行测试。

```
QEMU
Alloc memBlock E, failed!
dPartition(start=0x105020, totalN=0x4, perSize=0x20, firstFree=0x7f88144)
EBD(start=0x7f88144, next=0x0)
EBD(start=0x0, next=0xf000c2c3)
EBD(start=0xf000c2c3, next=0x0)
Now, release A.
cFPartition(start=0x105020, totalN=0x4, perSize=0x20, firstFree=0x10502c)
EBD(start=0x10502c, next=0x7f88144)
EBD(start=0x7f88144, next=0x0)
EBD(start=0x0, next=0xf000c2c3)
EBD(start=0xf000c2c3, next=0x0)
Now, release B.
dPartition(start=0x105020, totalN=0x4, perSize=0x20, firstFree=0x10504c)
EBD(start=0x10504c, next=0xbbbbbbbb)
EBD(start=0xbbbbbbbb, next=0x0)
Now, release C.
cFPartition(start=0x105020, totalN=0x4, perSize=0x20, firstFree=0x10506c)
EBD(start=0x10506c, next=0xc0000000)
EBD(start=0xc0000000, next=0x0)
Now, release D.
cFPartition(start=0x105020, totalN=0x4, perSize=0x20, firstFree=0x10508c)
EBD(start=0x10508c, next=0xd0000000)
EBD(start=0xd0000000, next=0x0)
huyixiang %
Unknown interrupt
```

图 17

输入 testkMalloc1，进行测试。

```
QEMU
EBD(start=0xd0000000, next=0x0)
huyixiang %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.
maxMallocSizeNow: Max_MemLOC_SIZE always changes. What's the value Now?
testdP1: Init a dPartition(size=0x100). [Alloc,Free] with step = 0x20
testdP2: Init a dPartition(size=0x100). A:B:C:- ==> -:B:C:- ==> -:C:- ==> -
testdP3: Init a dPartition(size=0x100). A:B:C:- ==> A:B:- ==> A:- ==> -
testeFP: Init a cFPartition. Alloc all and Free all.
testkMalloc1: kMalloc, write and read.
end
huyixiang %:testM
UNKNOWN command: testM
huyixiang %:testkMalloc1
16 allocated 2 buffers.
BUF1(size=9, addr=0x10503c) filled with 9(9): *****
BUF2(size=19, addr=0x105050) filled with 19(1): *****
huyixiang %:
Unknown interrupt
```

图 18

遇到的问题 and 解决方案说明

1. 编译时出现报错“对 ‘_GLOBAL_OFFSET_TABLE_’ 未定义的引用”。
解决方案: 在 src 目录下的 Makefile 文件的 CFLAGS 变量中添加 -fno-pic.
2. 编译出现 fatal error: bits/libc-header-start.h: No such file or directory
解决方案: 在 Ubuntu 中输入 apt-get install gcc-multilib, 完善编译环境
3. 编译时出现 warning: assignment makes pointer from integer without a cast
解决方案: 在给指针赋值前进行强制格式转化(unsigned short int*)
4. 编译时出现 warning: conflicting types for built-in function
解决方案: 在 src 目录下的 Makefile 文件的 CFLAGS 变量中添加 -fno-builtin