**实验四, 内存管理**

**李卓 pb19000064**

**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。

**实验内容**

内存检测，确定动态内存的范围

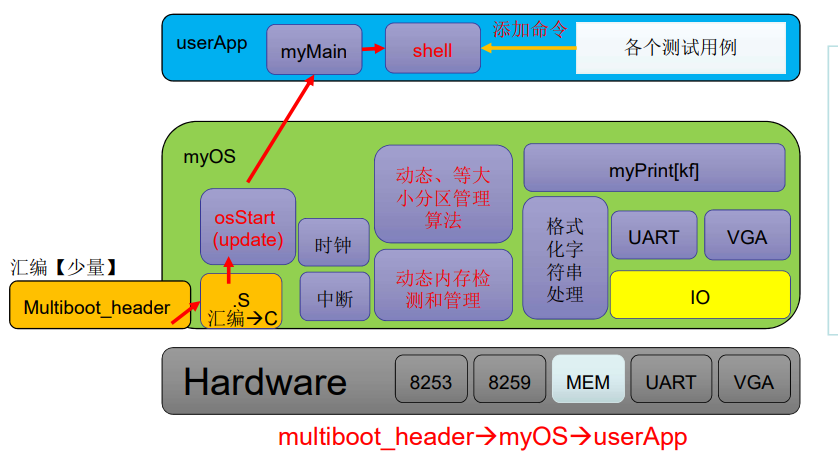
提供动态分区管理机制dPartition

提供等大小固定分区管理机制ePartition

使用动态分区管理机制来管理所有动态内存

提供kmalloc/kfree和malloc/free两套接口，分别 提供给内核和用户

**实验框架**



内核：内存（算法、检测、管理） 用户：新功能测试

被测功能：动态内存管理功能、两种算法

自测：userApp 他测：替换userApp或增加shell命令

**实验流程**



1. 在 multiboot\_header 中完成系统的启动。

2. 在 start32.S 中做好准备，调用 osStart.c 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

**实验原理**

1.内存检查

设从1M开始–,假设只有1块连续的内存空间, 通过检测得到这个内存块的大小

检测算法 void memTest(unsigned long start, unsigned long grainSize)

• 从start开始，以grainSize为步长，进行内存检测

• 检测方法： 1）读出grain的头2个字节

2）覆盖写入0xAA55，再读出并检查是否是0xAA55，若不是则检测结束；

3）覆盖写入0x55AA，再读出并检查是否是0x55AA，若不是则检测结束；

4）写回原来的值

5）对grain的尾2个字节，重复2-4

6）步进到下一个grain，重复1-5，直到检测结束

2. 动态分区管理

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

3. 等大小分区管理

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

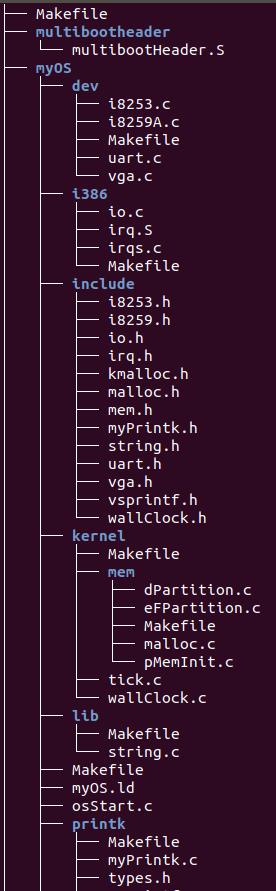
eFPartitionInit()实现将所有块按顺序连接成链，便于分配和回收。

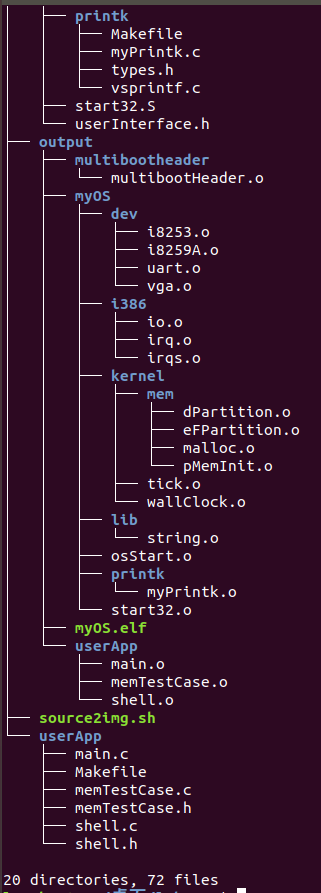
eFPartitionAlloc()实现块的分配，对链表进行重新链接，返回供使用的地址。 eFPartitionFree()实现块的逐一回收并重新链接至链表上。

4. addNewCmd

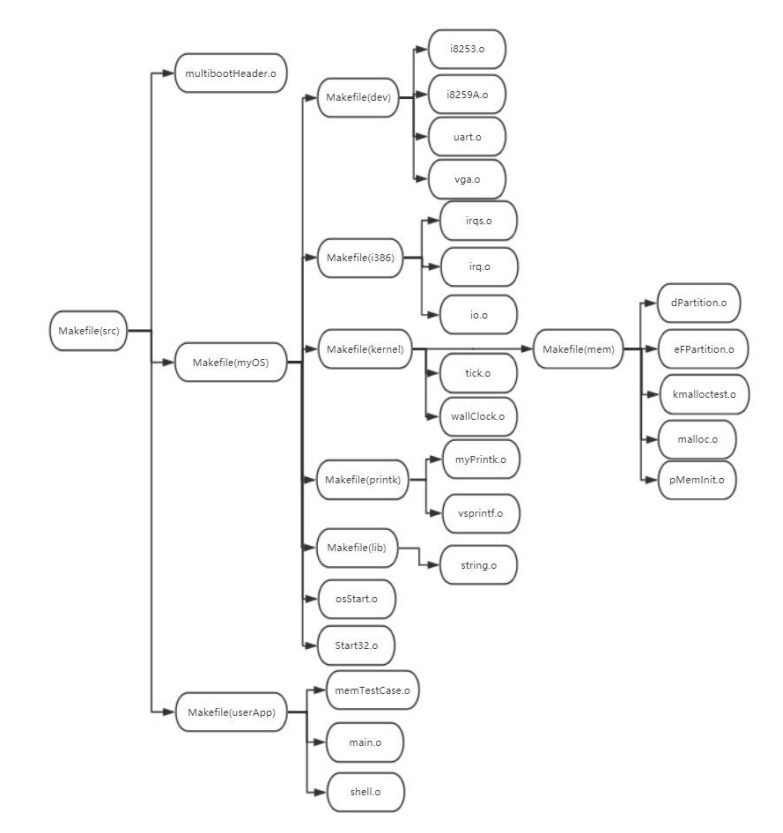
维护一个cmd链表, 是实现 shell 中的命令的动态添加

**文件目录组织**





makefile组织



**地址空间布局**

|  |  |  |
| --- | --- | --- |
| Section | Offset (Base = 1M) | align |
| .multiboot\_header | 0 | 8 |
| .text(代码段) | 16 | 8 |
| .data(数据段) | 16+.text section | 16 |
| .bss | 当前 | 16 |
| 堆栈(动态内存空间) | 当前 |  |

**编译过程说明**

默认方式, 链接生成 myOS.elf 文件

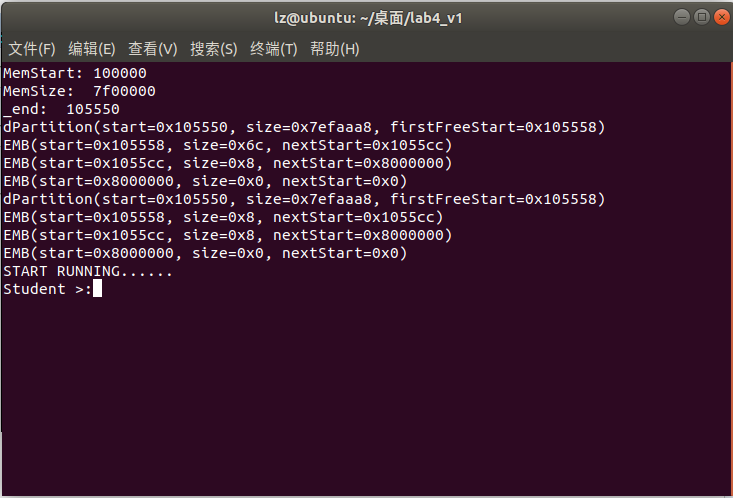
chmod 777 source2run.sh

./source2run.sh

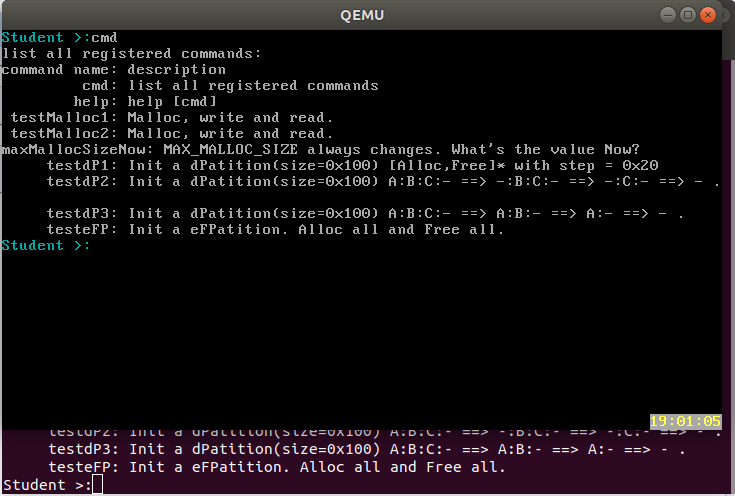
sudo screen /dev/pts/1

**运行结果**

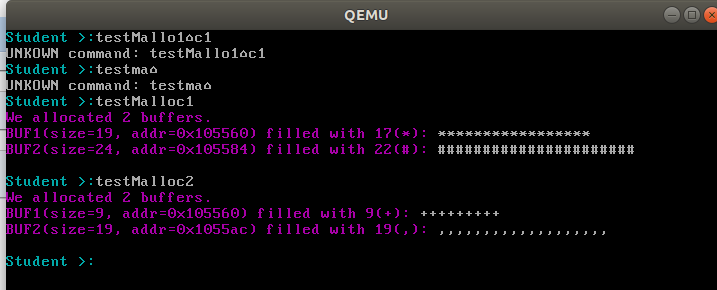




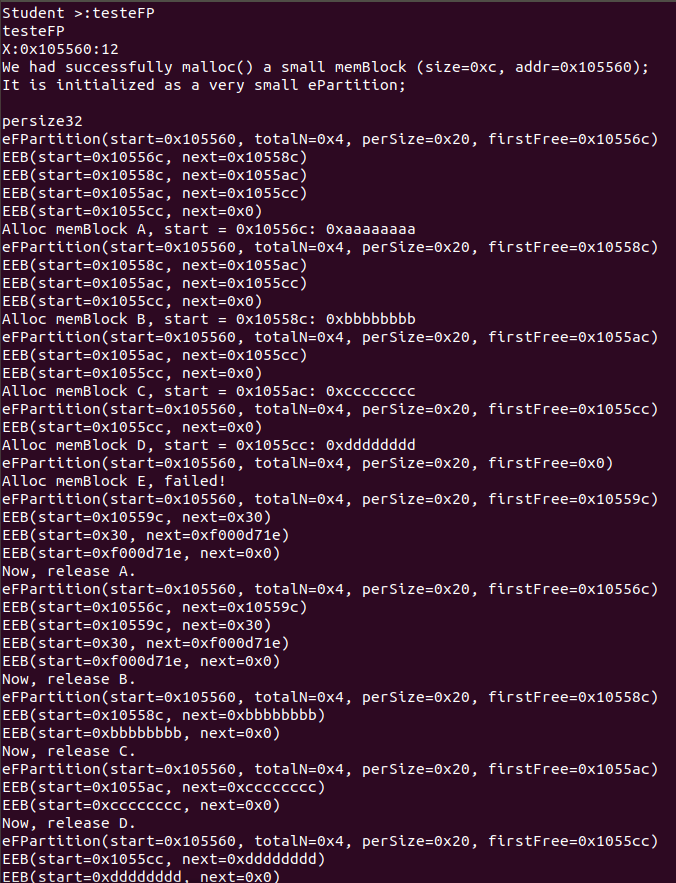
cmd



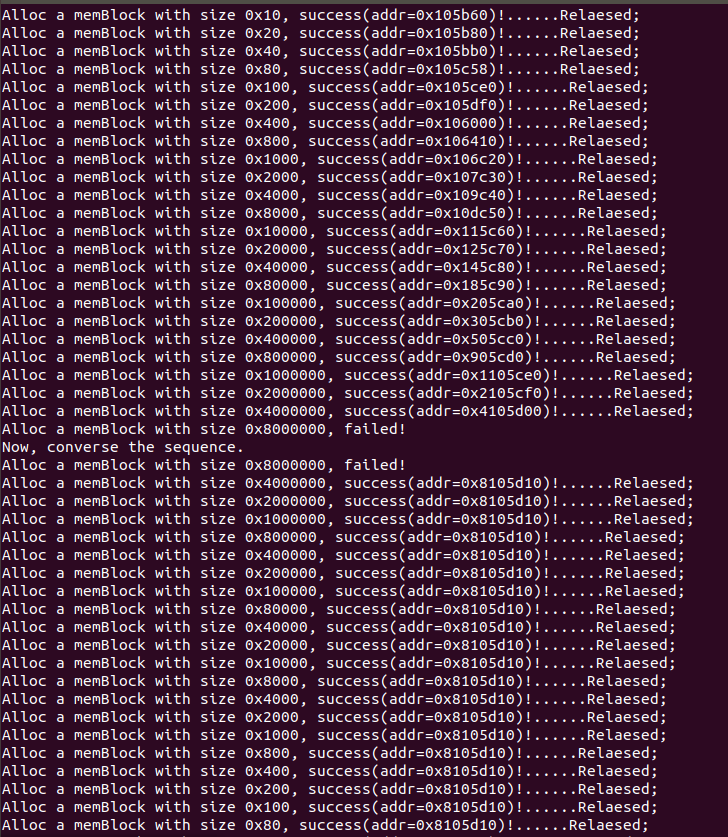
testMalloc1 testMalloc2



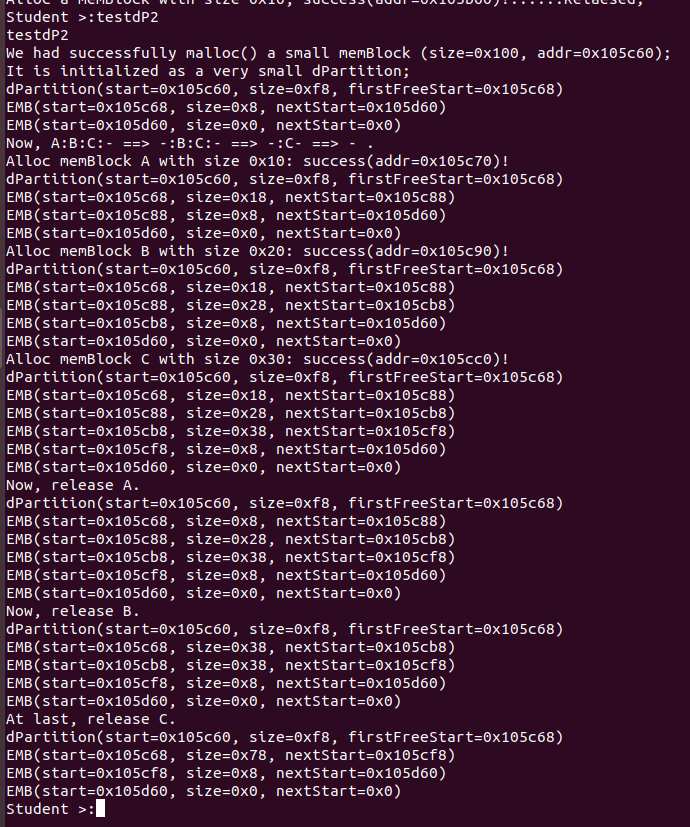
testeFP



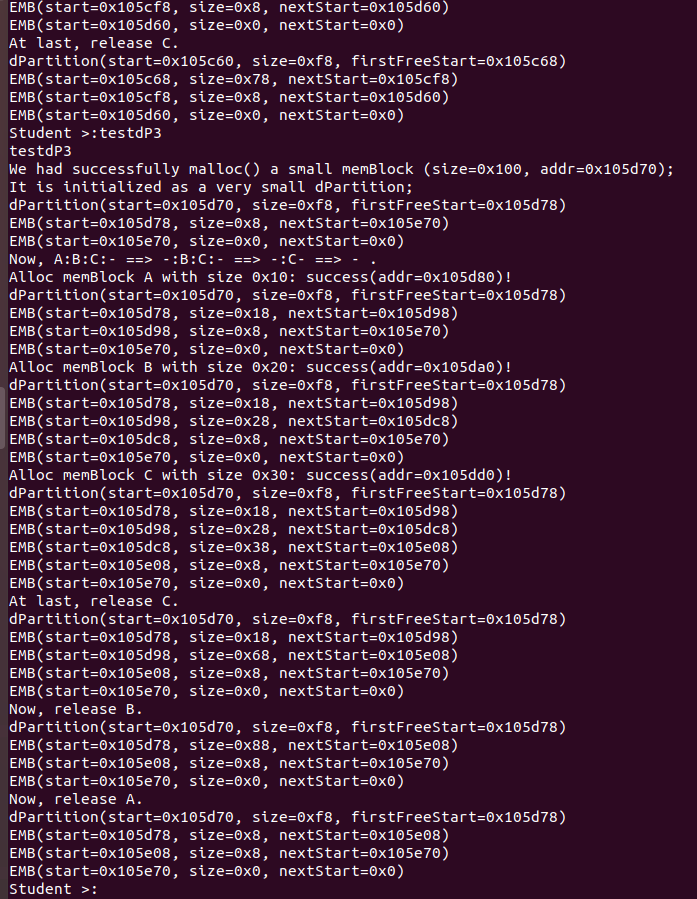
testdP1



testdP2



testdP3



maxMallocSizeNow

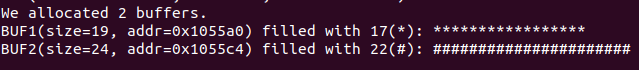


**运行结果解释**

*testMalloc1:*

malloc两块长为19和24的空间,分别用\* 和#填充, 然后free,.

由于firstfit算法, 此时buffer1 和buffer2空间上是连续的. buffer1逻辑长度为19 但实际长度计算为0x24, 因为对齐需要，EMB占位　和隔离带．可以看到, 逻辑地址在emb实际地址8位后



EMB如图



*testMalloc2:*

和testMalloc1同理



占用的EMB如图



实际空间位0x14 和0x20

释放后EMB如图



得到一个0x30大小的空间

*maxMallocSizeNow:*

步长0x1000



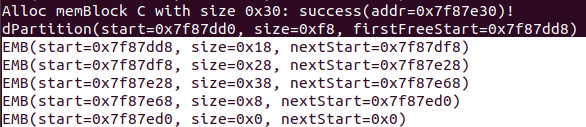
*testdP1;*

从0x10开始,步长不断翻倍 申请空间, 直到失败,全部释放

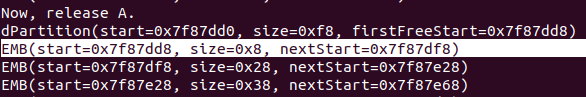


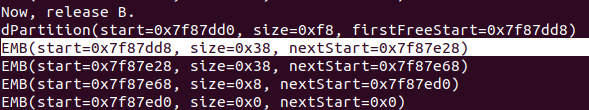
最后一次申请, 大小0x4000000, 最后的地址为0x8007cf0

*testdP2:*

A B C 全部申请后,如图,结果正确  


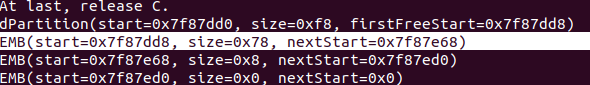
释放A后



释放B后:  


可以看到,A和B空闲的空间是连续的,所以被合并了,新空间大小为0x10+0x20+0x08

结果正确

释放C后:  


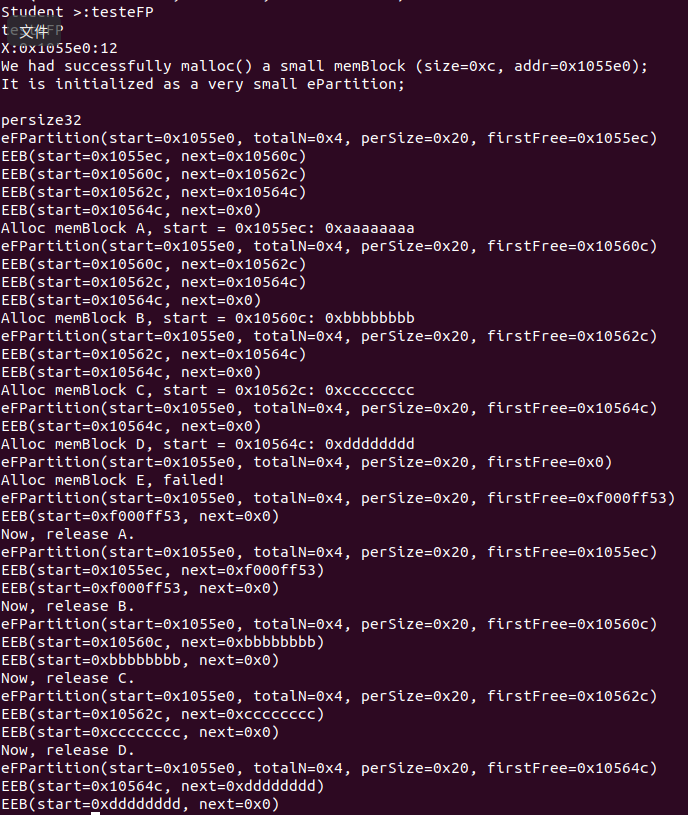
同理,空闲空间被合并

*testdP3:*

和testdP2同理

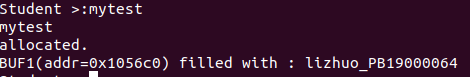
*testeFP*

如图所示, ABCD都申请成功，E失败，结果正确



*mytest:*

自己设计的样例, 输出自己的姓名和学号, 调用addNewCmd添加到cmd中



**实验中遇到的问题**

1. 没理清文件结构, 对全局变量重定义
2. 使用指针前,忘记判断是否为空指针
3. 对size理解出错, 实际空间比 逻辑空间大