Lab4 实验报告

毛陈诚 PB20111694

一、实验目标

- 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

二、源代码说明

对一些关键的设计进行说明，比如如何维护空闲链表，如何在释放后合并空间

维护空闲链表

malloc

将已分配块的后继作为已分配块的前驱的后继节点，将已分配块的空闲指针置0

```
    unsigned long EMBptr = dPartition->firstFreeStart;
    unsigned long preEMBptr = dp;
    int flag = 0; // flag用于标记空闲块是否是第一个空闲块
    //first-fit，从低到高寻找
    for (;; preEMBptr = EMBptr, EMBptr = ((EMB *)EMBptr)->nextStart)
    {
        if (EMBptr == END)
        {
            //myPrintk(0x7, "no suitable space for size %ld\n", size);
            return 0;
        }
        if (((EMB *)EMBptr)->size >= size){
            if(EMBptr == dPartition->firstFreeStart) flag = 1;
            break;
        }
    }
    unsigned long allocat_addr = (unsigned long)EMBptr;
    //如果产生新的空间
    if (((EMB *)EMBptr)->size - size > MIN_MEMEORY)
    {
        unsigned long newEMBptr = (unsigned long)(EMBptr) + size + EMB_size;
        if (flag == 1)
        {
            dPartition->firstFreeStart = newEMBptr;
        }
        if (flag == 0)
```

```
        {
            ((EMB *)preEMBptr)->nextStart = newEMBptr;
        }
        ((EMB *)newEMBptr)->nextStart = ((EMB *)EMBptr)->nextStart;
        ((EMB *)newEMBptr)->size = ((EMB *)EMBptr)->size - size - EMB_size;
    }
    //如果没有产生新的空闲区间，
    else if (((EMB *)EMBptr)->size - size <= MIN_MEMEORY)
    {
        if (flag == 1)
        {
            dPartition->firstFreeStart = ((EMB *)EMBptr)->nextStart;
        }
        if (flag == 0)
        {
            ((EMB *)preEMBptr)->nextStart = ((EMB *)EMBptr)->nextStart;
        }
    }
    //将已分配块的链表指针置0
    ((EMB *)EMBptr)->size = size;
    ((EMB *)EMBptr)->nextStart = 0;
```

free

遍历空闲链表，将新产生的空闲块插入合适的位置

```
    unsigned long end = start + ((EMB *)start)->size;
    //检查要释放的start~end这个范围是否在dp有效分配范围内
    if (end > ((dPartition *)dp)->size + dp | start < dp + dPartition_size){
        //myPrintk(0x7, "end:%d right:%d",end,((dPartition *)dp)->size + dp);
        //myPrintk(0x7, "start:%d left:%d",start,(dp + dPartition_size));
        return 0;
    }
    ((EMB *)start)->nextStart = END; //

    dPartition *dPartition = dp;
    unsigned long EMBptr = dPartition->firstFreeStart;
    unsigned long followEMBptr = ((EMB *)EMBptr)->nextStart;
    //如果是空闲链表为空
    if (dPartition->firstFreeStart == END)
        dPartition->firstFreeStart = start;
    //如果空闲块将插入空闲链表头部
    else if (dPartition->firstFreeStart > start)
    {
        ((EMB *)start)->nextStart = dPartition->firstFreeStart;
        dPartition->firstFreeStart = start;
        //合并空闲块
        Intersect(start, ((EMB *)start)->nextStart);
    }
    //如果空闲块将插入空闲链表中部或尾部
    else
```

```
    {
        for (;;)
        {//遍历空闲链表
            if (followEMBptr == END)
                break;
            if (EMBptr < start && followEMBptr > start){
                ((EMB *)start)->nextStart = followEMBptr;
                ((EMB *)EMBptr)->nextStart = start;
                break;
            }
            EMBptr = ((EMB *)EMBptr)->nextStart;
            followEMBptr = ((EMB *)EMBptr)->nextStart;
        }
        //合并空闲块
        Intersect(start, followEMBptr);
        Intersect(EMBptr, start);
    }
```

释放后合并空间

如果空闲块前后相接，需要进行合并。该函数用于上述的`free`函数

```
void Intersect(unsigned long start1, unsigned long start2)
{
    if (start2 == END)
        return;
    if (((EMB *)start1)->size + start1 + EMB_size == start2)
    {
        ((EMB *)start1)->size += ((EMB *)start2)->size + EMB_size;
        ((EMB *)start1)->nextStart = ((EMB *)start2)->nextStart;
    }
}
```

三、问题回答

1

`malloc`调用了`dPartitionAlloc`，`dPartitionAlloc`调用了`dPartitionAllocFirstFit`

2

`cmd`

```
Student >:cmd
cmd
list all registered commands:
command name: description
     testeFP: Init a eFPatition. Alloc all and Free all.
     testdP3: Init a dPatition(size=0x100) A:B:C:- ==> A:B:- ==> A:- ==>
- .
     testdP2: Init a dPatition(size=0x100) A:B:C:- ==> -:B:C:- ==> -:C:-
==> - .
     testdP1: Init a dPatition(size=0x100) [Alloc,Free]* with step = 0x20
maxMallocSizeNow: MAX_MALLOC_SIZE always changes. What's the value Now?
 testMalloc2: Malloc, write and read.
 testMalloc1: Malloc, write and read.
        help: help [cmd]
         cmd: list all registered commands
```

`testMalloc1`与`testMalloc2`

输入与输出一致

```
Student >:testMalloc1
testMalloc1
We allocated 2 buffers.
BUF1(size=19, addr=0x105c78) filled with 17(*): ******************
BUF2(size=24, addr=0x105c94) filled with 22(#): ######################

Student >:testMalloc2
testMalloc2
We allocated 2 buffers.
BUF1(size=9, addr=0x105cb4) filled with 9(+): +++++++++
BUF2(size=19, addr=0x105cc8) filled with 19(,): ,,,,,,,,,,,,,,,,,,,
```

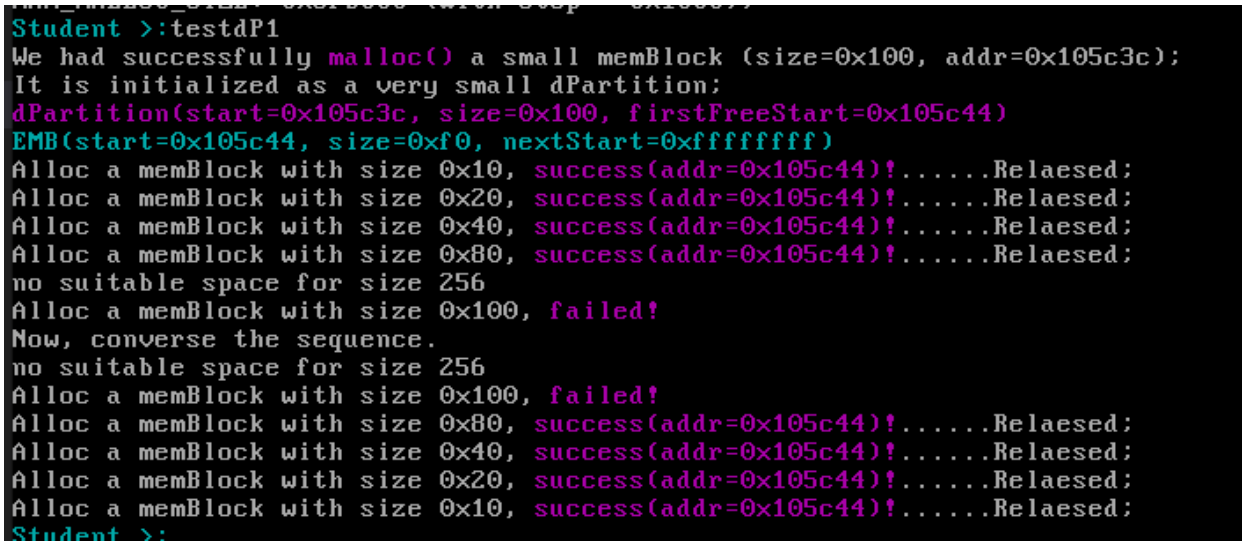
`maxMallocSizeNow`

最大分配空间0xefb000

```
Student >:maxMallocSizeNow
maxMallocSizeNow
no suitable space for size 15708160
MAX_MALLOC_SIZE: 0xefb000 (with step = 0x1000);
```

`testdp1`

分配0x10到0x80成功，分配0x100失败。因为这个分区总共只有0x100大小，除去dpartition和EMB大小只有0x84可用空间



`testd2`

一次分配3个块，所以EMB从一个到四个(包含未分配的一个块)

在释放过程中，释放A，导致A的分配块变成空闲块，仍有四个块，

释放B，A和B的空闲块合并，变成三个块

释放C，A,B,C的空闲块和原空闲块合并，变成1个块

```
We had successfully malloc() a small memBlock (size=0x100, addr=0x105c3c)
;
It is initialized as a very small dPartition;
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c44)
EMB(start=0x105c44, size=0xf0, nextStart=0xffffffff)
Now, A:B:C:- ==> -:B:C:- ==> -:C- ==> - .
Alloc memBlock A with size 0x10: success(addr=0x105c44)!
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c5c)
EMB(start=0x105c44, size=0x10, nextStart=0x0)
EMB(start=0x105c5c, size=0xd8, nextStart=0xffffffff)
Alloc memBlock B with size 0x20: success(addr=0x105c5c)!
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c84)
EMB(start=0x105c44, size=0x10, nextStart=0x0)
EMB(start=0x105c5c, size=0x20, nextStart=0x0)
EMB(start=0x105c84, size=0xb0, nextStart=0xffffffff)
Alloc memBlock C with size 0x30: success(addr=0x105c84)!
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105cbc)
EMB(start=0x105c44, size=0x10, nextStart=0x0)
EMB(start=0x105c5c, size=0x20, nextStart=0x0)
EMB(start=0x105c84, size=0x30, nextStart=0x0)
EMB(start=0x105cbc, size=0x78, nextStart=0xffffffff)
Now, release A.
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c44)
EMB(start=0x105c44, size=0x10, nextStart=0x105cbc)
EMB(start=0x105c5c, size=0x20, nextStart=0x0)
EMB(start=0x105c84, size=0x30, nextStart=0x0)
EMB(start=0x105cbc, size=0x78, nextStart=0xffffffff)
Now, release B.
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c44)
EMB(start=0x105c44, size=0x38, nextStart=0x105cbc)
EMB(start=0x105c84, size=0x30, nextStart=0x0)
EMB(start=0x105cbc, size=0x78, nextStart=0xffffffff)
At last, release C.
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c44)
EMB(start=0x105c44, size=0xf0, nextStart=0xfffff
```

`testdp3`

分配过程同上

释放C，C与空闲块合并成一个块，现有3个块

释放B，B,C与空闲块合并成一个块，现有2个块

释放A，A,B,C与空闲块合并成一个块，现有1个块

```
testdP3
We had successfully malloc() a small memBlock (size=0x100, addr=0x105c3c)
;
It is initialized as a very small dPartition;
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c44)
EMB(start=0x105c44, size=0xf0, nextStart=0xffffffff)
Now, A:B:C:- ==> -:B:C:- ==> -:C- ==> - .
Alloc memBlock A with size 0x10: success(addr=0x105c44)!
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c5c)
EMB(start=0x105c44, size=0x10, nextStart=0x0)
EMB(start=0x105c5c, size=0xd8, nextStart=0xffffffff)
Alloc memBlock B with size 0x20: success(addr=0x105c5c)!
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c84)
EMB(start=0x105c44, size=0x10, nextStart=0x0)
EMB(start=0x105c5c, size=0x20, nextStart=0x0)
EMB(start=0x105c84, size=0xb0, nextStart=0xffffffff)
Alloc memBlock C with size 0x30: success(addr=0x105c84)!
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105cbc)
EMB(start=0x105c44, size=0x10, nextStart=0x0)
EMB(start=0x105c5c, size=0x20, nextStart=0x0)
EMB(start=0x105c84, size=0x30, nextStart=0x0)
EMB(start=0x105cbc, size=0x78, nextStart=0xffffffff)
At last, release C.
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c84)
EMB(start=0x105c44, size=0x10, nextStart=0x0)
EMB(start=0x105c5c, size=0x20, nextStart=0x0)
EMB(start=0x105c84, size=0xb0, nextStart=0xffffffff)
Now, release B.
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c5c)
EMB(start=0x105c44, size=0x10, nextStart=0x0)
EMB(start=0x105c5c, size=0xd8, nextStart=0xffffffff)
Now, release A.
dPartition(start=0x105c3c, size=0x100, firstFreeStart=0x105c44)
EMB(start=0x105c44, size=0xf0, nextStart=0xffffffff)
```