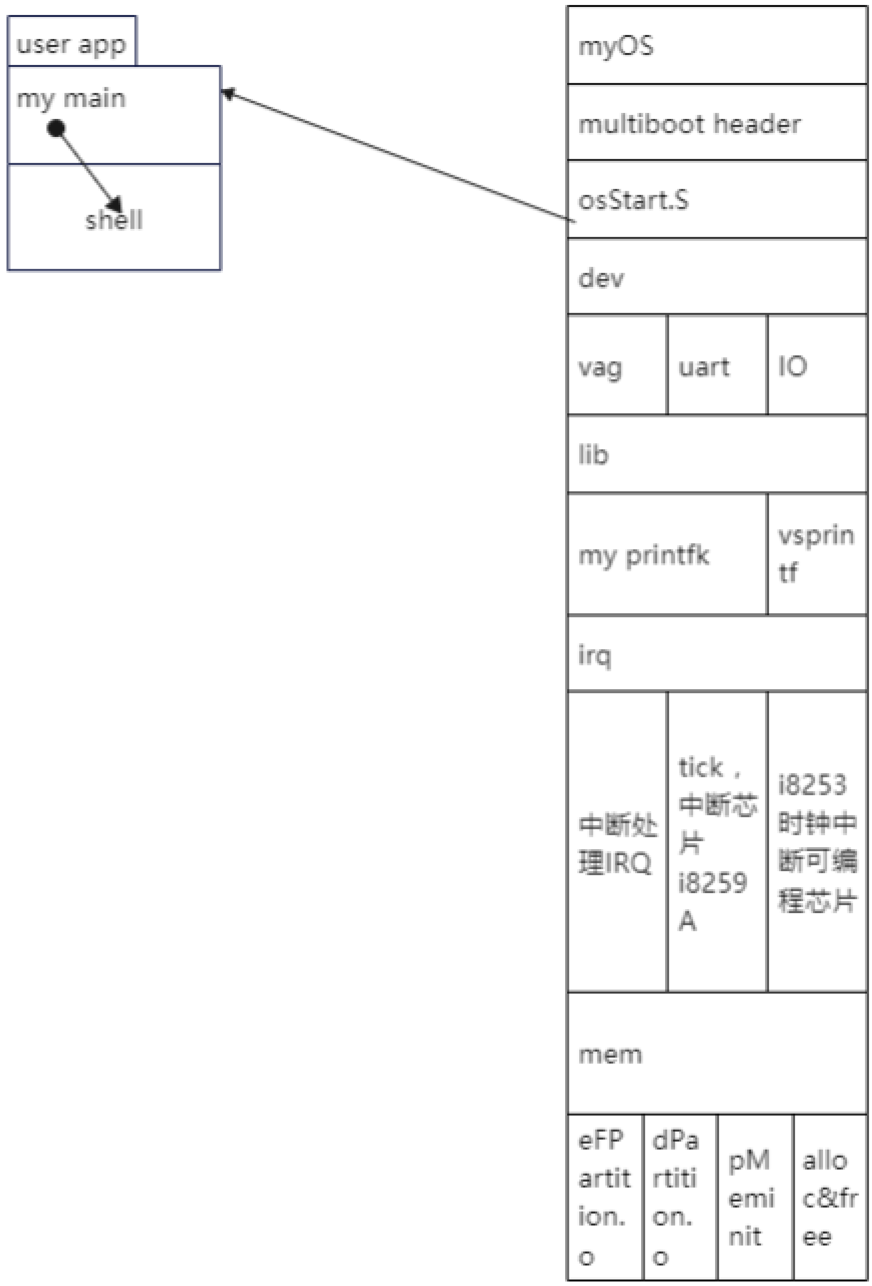
lab4

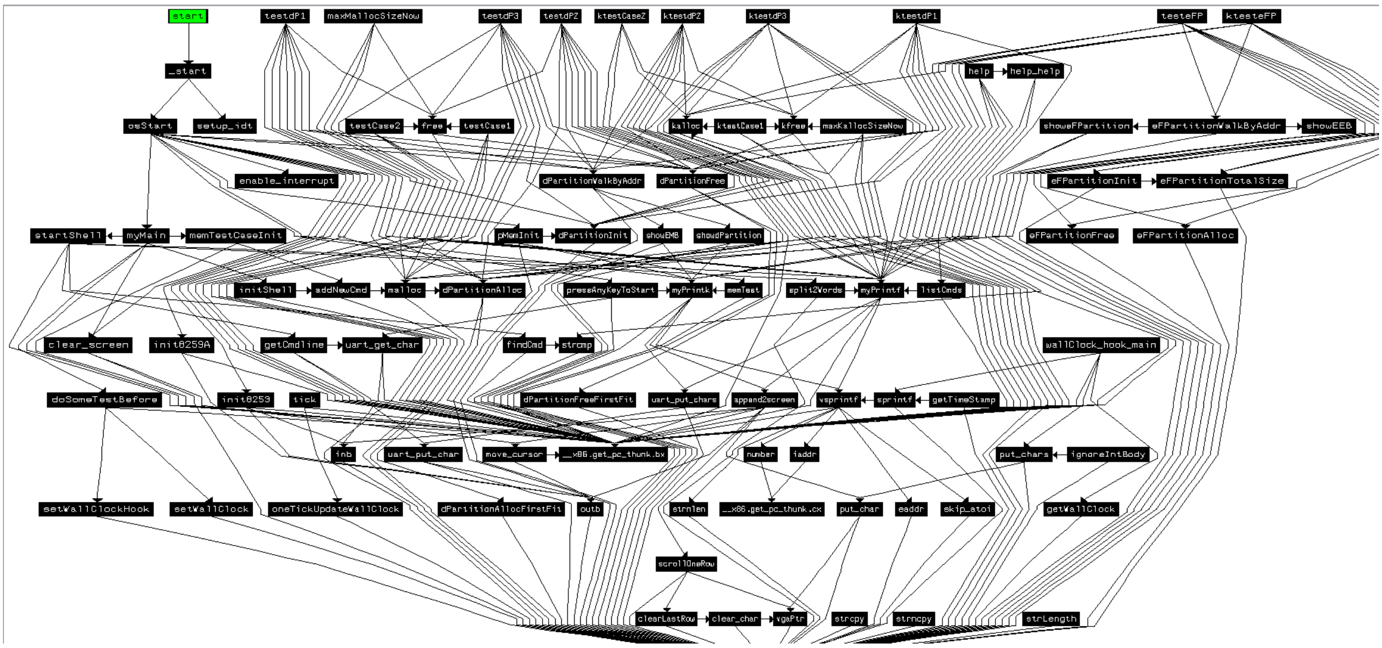
[TOC]

软件框图

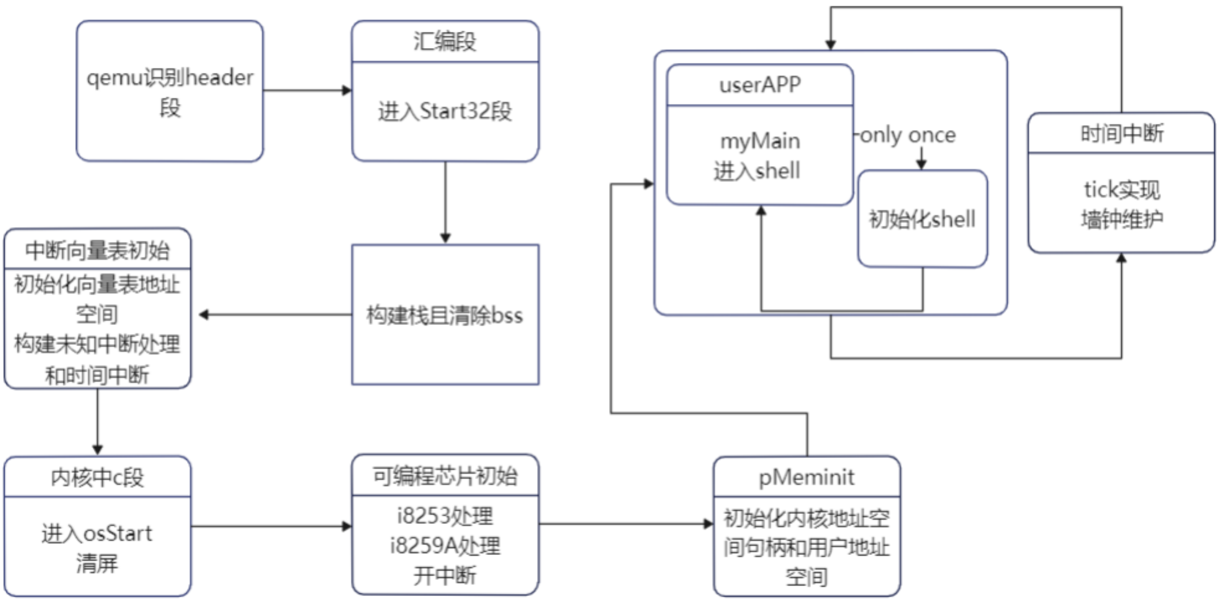


流程实现

函数关系



流程图



主要功能模块实现

dpartition

```
unsigned int dPartitionAllocFirstFit(unsigned int dp, unsigned int size)
{
    EMB *emb, *pre, *cur;
    unsigned int alignS;
    emb = *(EMB **)(dp + 4);
```

```
    if(!emb) return 0;

    // 4字节对齐
    alignS = size % 4 ? (size / 4 * 4 + 8) : size + 4;


    if (alignS + 8 > *(unsigned int *)dp)
        {return 0;}//如果内存过大return
    while (emb->size < (alignS - 4))
    {
        pre = emb;
        emb = (EMB *)emb->nextStart;
        if (!emb) return 0; //未找到满足要求的EMB
    }

    //第一块满足
    if (emb == *(EMB **)(dp + 4))
    {
        if (emb->size == (alignS - 4))
        {
            *(unsigned int *)dp = -emb->nextStart + *(unsigned int *)(dp + 4) + *
(unsigned int *)dp;
            (*(unsigned int *)(dp + 4)) = emb->nextStart;
        }
        else
        {
            *(unsigned int *)dp = *(unsigned int*)dp-alignS;
            (*(unsigned int *)(dp + 4)) += alignS;
            cur = *(EMB **)(dp + 4);
            cur->size = emb->size - alignS;
            cur->nextStart = emb->nextStart;
        }
    }

    //EMB正好满足
    else if ((alignS - 4) == emb->size)
    {
        pre->nextStart = emb->nextStart;
    }

    //一般
    else
    {
        pre->nextStart = (unsigned int)emb + alignS;
        *(unsigned int *)(pre->nextStart) = emb->size - alignS;
        *(unsigned int *)(pre->nextStart + 4) = emb->nextStart;
    }

    emb->size = alignS - 4;
    return (unsigned int)emb + 4;
}

unsigned int dPartitionFreeFirstFit(unsigned int dp, unsigned int start)
```

```
{
    start -= 4;
    unsigned int end = start + (*(unsigned int *)start) + 4;
    //边界条件
    if (end >= *(unsigned int *)dp + *(unsigned int *)(dp+4)-8 ||  start < dp + 8)
        return 0;

    //第一块空闲块前
    if (end < *(unsigned int *)(dp + 4))
    {
        *(unsigned int *)(start + 4) = *(unsigned int *)(dp + 4);
        *(unsigned int *)dp+=(*(unsigned int *)(dp+4)-start);
        *(unsigned int *)(dp + 4) = start;
        return 1;
    }
    if (end == *(unsigned int *)(dp + 4))
    {
        *(unsigned int *)(dp + 4) = start;
        *(unsigned int *)dp +=*(unsigned int *) start+4;
        *(unsigned int *)start = *(unsigned int *)end + 4 + *(unsigned int *)start;
        *(unsigned int *)(start + 4) = *(unsigned int *)(end + 4);

        return 1;
    }



    EMB *pre;
    EMB *emb = *(EMB **)(dp + 4);
    //遍历找到释放块前后块
    while (start > (unsigned int)emb && (unsigned int)emb)
    {
        pre = emb;
        emb = (EMB *)(emb->nextStart);
    }

    //被释放为最后一块
    if (emb == 0)
    {
        if ((unsigned int)pre + pre->size + 4 == start)
        {
            pre->size = pre->size + 4 + *(unsigned int *)start;
            return 1;
        }
        else
        {
            pre->nextStart = start;
            *(unsigned int *)(start + 4) = 0;
        }
    }


    else
```

```
    {
        int flag = 0;
        //基于flag实现块的链接，寻找时只需进行合并操作

        //integrate
        if ((unsigned int)emb == end)//2-3
        {
            (*(unsigned int *)start) += (4 + emb->size);
            flag += 1;
        }
        if ((unsigned int)pre + pre->size + 4 == start)//1-2
        {
            pre->size = pre->size + 4 + *(unsigned int *)start;
            flag += 2;
        }
        //link
        switch (flag)
        {
        case 1://2-3
            *(unsigned int *)(start + 4) = emb->nextStart;
            break;
        case 2://1-2
            break;
        case 3://1-2-3
            pre->nextStart = emb->nextStart;
            break;
        default:
            pre->nextStart = start;
            *(EMB **)(start + 4) = emb;
            break;
        }
    }
    return 1;
}
//返回的地址统一是可以直接使用的初始地址
```

efpartition

```
unsigned int eFPartitionTotalSize(unsigned int perSize, unsigned int n){
    return n*(perSize+4)+eFPartition_size;
}

unsigned int eFPartitionInit(unsigned int start, unsigned int perSize, unsigned int n){
    //4字节对齐
    perSize=(perSize+3)/4*4;
    //初始化句柄
    *(unsigned int *)start=eFPartitionTotalSize(perSize,n);
    *(unsigned int *)(start+4)=perSize;

    //初始化EEB描述字
    *(unsigned int *)(start+8)=start+12;
```

```
    for(int i=0;i<n-1;i++){
        *(unsigned int *)(start+12+i*(perSize+4))=start+12+(i+1)*(perSize+4);
    }
    return start;
}

unsigned int  eFPartitionAlloc(unsigned int EFPHandler)
{
  EEB *eeb;
  if ( ! *(unsigned int *)(EFPHandler + 8) )return 0;//如果句柄无效return

  eeb = *(EEB **)(EFPHandler + 8);
  *(unsigned int *)(EFPHandler + 8) = eeb->next_start;
  return (unsigned int)eeb +4;
}


unsigned int eFPartitionFree(unsigned int EFPHandler,unsigned int mbStart){
if(mbStart<EFPHandler+16)return 0;//如果越界 return
int a=((mbStart-4)-EFPHandler-12)%(*(unsigned int *)(EFPHandler+4)+4);
if(a) return 0;//判断start地址是否正确（EEB头地址）
*(unsigned int *)(mbStart-4)=*(unsigned int *)(EFPHandler+8);
*(unsigned int *)(EFPHandler+8)=mbStart-4;
return 1;
}
```

pMeminit

```
void memTest(unsigned long start, unsigned long grainSize){
    unsigned short *p;
    unsigned short pdata;
    unsigned long addr;
    addr = start;
    pMemStart = start;
    pMemSize = 0;

    if (start < 0x100000 || grainSize < 4) return;  //检测是否越界和grainsize是否过小
    while (1) {
        //检测首两个字节
        p = (unsigned  short*) addr;
        pdata = *p;
        *p = 0xAA55;
        if (*p != 0xAA55 ) break;
        *p = 0x55AA;
        if (*p != 0x55AA ) break;
        *p = pdata;//复原
        //检测末两个字节
        p = (unsigned  short*) (addr + grainSize - 2);
        pdata = *p;
        *p = 0xAA55;
        if (*p != 0xAA55 ) break;
```

```
        *p = 0x55AA;
        if (*p != 0x55AA ) break;
        *p = pdata;//复原
        //下一块
        pMemSize += grainSize;
        addr += grainSize;
    }

    myPrintk(0x7,"MemStart:@ %x  \n", pMemStart);
    myPrintk(0x7,"MemSize@  %x  \n", pMemSize);
}

extern unsigned int _end;
void pMemInit(void)
{
  unsigned int _end_addr = (unsigned int)&_end;

  memTest(0x100000, 0x1000);
  myPrintk(0x7, "_end:  %x  \n", _end_addr);
  if (pMemStart <= _end_addr)
  {
    //init kernel
    kstart = _end_addr;
    kMemSize=uMemS-kstart;
    //init user
    ustart=uMemS;
    uMemSize=pMemSize-kMemSize;
  }
    //init kernel
  kMemHandler = dPartitionInit(kstart, kMemSize);
    //init user
  pMemHandler = dPartitionInit(ustart,uMemSize);
}
```

addnewcmd

```
void addNewCmd( unsigned char *Cmd,
        int (*func)(int argc, unsigned char **argv),
        void (*help_func)(void),
        unsigned char* description){
  cmd *newCmd;
  cmd *tmpCmd;
  int i;
  int ia;
    //分配地址空间
  newCmd = (cmd *)malloc(sizeof(cmd));

  i = -1;
  do
  {
    i++;
```

```
    newCmd->cmd[i] = Cmd[i];
  }
  while ( Cmd[i] && i <= 20 );


  ia = 0;
  do
  {
    newCmd->description[ia] = description[ia];
    ia++;
  }
  while ( description[ia] && ia <= 100 );
  newCmd->func = func;
  newCmd->help_func = help_func;
  newCmd->nextCmd = 0;
  if ( ourCmds )
  {
    for ( tmpCmd = ourCmds; tmpCmd->nextCmd; tmpCmd = tmpCmd->nextCmd ){;};
    //利用for 循环遍历链表
    tmpCmd->nextCmd = newCmd;
  }
  else
  // init
  {
    ourCmds = newCmd;
  }
}
```

关于kmem 和umem实现

总共大小约 2^7M 用户与内核均分 设置user 初始地址$0x4'000'000$

初始化shell 在用户区 并修改了相应头文件

```
newCmd = (cmd *)malloc(sizeof(cmd));
```

实现

> 地址分配
>
> - 独立维护两者的句柄 达到分离的目的
>
> ```
> #define uMemS 0x4000000
>  if (pMemStart <= _end_addr)
>  {
>    // pMemSize -= _end_addr - pMemStart;
>    // pMemStart=_end_addr;
>    kstart = _end_addr;
>    kMemSize=uMemS-kstart;
> ```

```
      ustart=uMemS;
      uMemSize=pMemSize-kMemSize;
    }
    // pMemHandler = dPartitionInit(pMemStart,pMemSize);
    kMemHandler = dPartitionInit(kstart, kMemSize);
    pMemHandler = dPartitionInit(ustart,uMemSize);
```

alloc

```
//user
unsigned long malloc(unsigned long size){
    return dPartitionAlloc(pMemHandler,size);
}
unsigned long free(unsigned long start){
    return dPartitionFree(pMemHandler,start);
}
//kernel
unsigned long kalloc(unsigned long size){
    return dPartitionAlloc(kMemHandler,size);
}
unsigned long kfree(unsigned long start){
    return dPartitionFree(kMemHandler,start);
}
```

测试用例

仅是将malloc 的替换 位kalloc free->kfree

包括shell指令 与OSstart运行时 测试

```
MemStart: 100000
MemSize:  7f00000
_end:  106ab0
dPartition(start=0x4000000, size=0x4006ab0, firstFreeStart=0x4000008)
EMB(start=0x4000008, size=0x4006aa4, nextStart=0x0)
dPartition(start=0x4000000, size=0x4006a48, firstFreeStart=0x4000070)
EMB(start=0x4000070, size=0x4006a3c, nextStart=0x0)
dPartition(start=0x4000000, size=0x4006ab0, firstFreeStart=0x4000008)
EMB(start=0x4000008, size=0x4006aa4, nextStart=0x0)
dPartition(start=0x106ab0, size=0x3ef9550, firstFreeStart=0x106ab8)
EMB(start=0x106ab8, size=0x3ef9544, nextStart=0x0)
dPartition(start=0x106ab0, size=0x3ef94e8, firstFreeStart=0x106b20)
EMB(start=0x106b20, size=0x3ef94dc, nextStart=0x0)
dPartition(start=0x106ab0, size=0x3ef9550, firstFreeStart=0x106ab8)
EMB(start=0x106ab8, size=0x3ef9544, nextStart=0x0)
```

```
maxkallocSizeNow: MAX_KALLOC_SIZE always changes. What's the value Now?
ktestMalloc1: kalloc, write and read.
ktestMalloc2: kalloc, write and read.
    ktestdP1: Init a dPatition(size=0x100) [Alloc,Free]* with step = 0x20
    ktestdP2: Init a dPatition(size=0x100) A:B:C:- ==> -:B:C:- ==> -:C:- ==> - .
    ktestdP3: Init a dPatition(size=0x100) A:B:C:- ==> A:B:- ==> A:- ==> - .
    ktesteFP: Init a eFPatition. Alloc all and Free all.
```

源代码组织

目录组织

```
.
├── Makefile
├── multibootheader
│   └── multibootHeader.S
├── myOS
│   ├── dev
│   │   ├── i8253.c
│   │   ├── i8259A.c
│   │   ├── Makefile
│   │   ├── uart.c
│   │   └── vga.c
│   ├── i386
│   │   ├── io.c
│   │   ├── irq.S
│   │   ├── irqs.c
│   │   └── Makefile
│   ├── include
│   │   ├── i8253.h
│   │   ├── i8259.h
│   │   ├── io.h
│   │   ├── irq.h
│   │   ├── kmalloc.h
│   │   ├── malloc.h
│   │   ├── mem.h
│   │   ├── myPrintk.h
│   │   ├── string.h
│   │   ├── uart.h
│   │   ├── vga.h
│   │   ├── vsprintf.h
│   │   └── wallClock.h
│   ├── kernel
│   │   ├── Makefile
│   │   ├── mem
│   │   │   ├── dPartition.c
│   │   │   ├── eFPartition.c
│   │   │   ├── Makefile
│   │   │   ├── malloc.c
│   │   │   └── pMemInit.c
```

```
│   │   ├── tick.c
│   │   └── wallClock.c
│   ├── lib
│   │   ├── Makefile
│   │   └── string.c
│   ├── Makefile
│   ├── myOS.ld
│   ├── osStart.c
│   ├── printk
│   │   ├── Makefile
│   │   ├── myPrintk.c
│   │   ├── types.h
│   │   └── vsprintf.c
│   ├── start32.S
│   └── userInterface.h
├── source2img.sh
├── tree
└── userApp
    ├── main.c
    ├── Makefile
    ├── memTestCase.c
    ├── memTestCase.h
    ├── shell.c
    └── shell.h

10 directories, 51 files
```

makefile 组织

```
.
├── multibootheader
│   └── multibootHeader.o
├── myOS
│   ├── dev
│   │   ├── i8253.o
│   │   ├── i8259A.o
│   │   ├── uart.o
│   │   └── vga.o
│   ├── i386
│   │   ├── io.o
│   │   ├── irq.o
│   │   └── irqs.o
│   ├── kernel
│   │   ├── mem
│   │   │   ├── dPartition.o
│   │   │   ├── eFPartition.o
│   │   │   ├── malloc.o
│   │   │   └── pMemInit.o
│   │   ├── tick.o
│   │   └── wallClock.o
```

```
|·· ├── lib
|·· |·· └── string.o
|·· ├── osStart.o
|·· ├── printk
|·· |·· └── myPrintk.o
|·· └── start32.o
├── myOS.elf
├── myOS.elf.id0
├── myOS.elf.id1
├── myOS.elf.id2
├── myOS.elf.nam
├── myOS.elf.til
└── userApp
    ├── main.o
    ├── memTestCase.o
    └── shell.o
```

地址空间

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class
.text	00100000	00104711	R	.	X	.	L	para	0002	public	CODE
LOAD	00104711	00104714	R	W	X	.	L	align_32	0001	public	CODE
.rodata	00104714	0010526A	R	.	.	.	L	dword	0003	public	CONST
LOAD	0010526A	0010526C	R	W	X	.	L	align_32	0001	public	CODE
.eh_frame	0010526C	00105E5C	R	.	.	.	L	dword	0004	public	CONST
.text.__x86.get_pc_thunk....	00105E5C	00105E60	R	.	X	.	L	byte	0005	public	CODE
.text.__x86.get_pc_thunk....	00105E60	00105E64	R	.	X	.	L	byte	0006	public	CODE
.text.__x86.get_pc_thunk....	00105E64	00105E68	R	.	X	.	L	byte	0007	public	CODE
LOAD	00105E68	00105E70	R	W	X	.	L	align_32	0001	public	CODE
.data	00105E70	00106680	R	W	.	.	L	para	0008	public	DATA
.got.plt	00106680	0010668C	R	W	.	.	L	dword	0009	public	DATA
LOAD	0010668C	001066A0	R	W	X	.	L	align_32	0001	public	CODE
.bss	001066A0	00106AA4	R	W	.	.	L	align_32	000A	public	BSS

> 后内核内存空间分配 0x106AB0 ---0x4'000'000
>
> 用户内核空间0x4'000'000

编译过程说明

编译所用指令

```
gcc -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector
gcc -m32 -fno-stack-protector -fno-builtin -g
```

编译大致过程

编译汇编文件和c语言文件形成可重定向的二进制文件，再通过ld 命令将可重定向文件重新组织链接形成elf文件

运行结果

用户测试

- testmalloc试验

```
    ktesteFP: Init a eFPatition. Alloc all and Free all.
Student >:maxMallocSizeNow
maxMallocSizeNow
MAX_MALLOC_SIZE: 0x4001000 (with step = 0x1000);
Student >:testMalloc1
testMalloc1
We allocated 2 buffers.
BUF1(size=19, addr=0x400090c) filled with 17(*): *****************
BUF2(size=24, addr=0x4000924) filled with 22(#): ######################

Student >:testMalloc2
testMalloc2
We allocated 2 buffers.
BUF1(size=9, addr=0x400090c) filled with 9(+): +++++++++
BUF2(size=19, addr=0x400091c) filled with 19(,): ,,,,,,,,,,,,,,,,,,,

Student >:maxMallocSizeNow
maxMallocSizeNow
MAX_MALLOC_SIZE: 0x4001000 (with step = 0x1000);
Student >:
```

> 成功输出且前后maxsize不变 内存成功分配且释放

- testdP1

```
We had successfully malloc() a small memBlock (size=0x100, addr=0x400090c);
It is initialized as a very small dPartition;
dPartition(start=0x400090c, size=0x100, firstFreeStart=0x4000914)
EMB(start=0x4000914, size=0xf4, nextStart=0x0)
Alloc a memBlock with size 0x10, success(addr=0x4000918)!......Relaesed;
Alloc a memBlock with size 0x20, success(addr=0x4000918)!......Relaesed;
Alloc a memBlock with size 0x40, success(addr=0x4000918)!......Relaesed;
Alloc a memBlock with size 0x80, success(addr=0x4000918)!......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=0x4000918)!......Relaesed;
Alloc a memBlock with size 0x40, success(addr=0x4000918)!......Relaesed;
Alloc a memBlock with size 0x20, success(addr=0x4000918)!......Relaesed;
Alloc a memBlock with size 0x10, success(addr=0x4000918)!......Relaesed;
```

> 在malloc申请的地址处成功init新的句柄进行子内存测试 申请0x100
>
> addr始终不变说明成功释放
>
> 0x100 未被分配 （虽然分配给子内存0x100 但由于句柄和内存块描述字 无法成功申请 ） 分配过程正常

- testdP2

```
dPartition(start=0x400090c, size=0x100, firstFreeStart=0x4000914)
EMB(start=0x4000914, size=0xf4, nextStart=0x0)
Now, A:B:C:- ==> -:B:C:- ==> -:C- ==> - .
Alloc memBlock A with size 0x10: success(addr=0x4000918)!
dPartition(start=0x400090c, size=0xec, firstFreeStart=0x4000928)
EMB(start=0x4000928, size=0xe0, nextStart=0x0)
Alloc memBlock B with size 0x20: success(addr=0x400092c)!
dPartition(start=0x400090c, size=0xc8, firstFreeStart=0x400094c)
EMB(start=0x400094c, size=0xbc, nextStart=0x0)
Alloc memBlock C with size 0x30: success(addr=0x4000950)!
dPartition(start=0x400090c, size=0x94, firstFreeStart=0x4000980)
EMB(start=0x4000980, size=0x88, nextStart=0x0)
Now, release A.
dPartition(start=0x400090c, size=0x100, firstFreeStart=0x4000914)
EMB(start=0x4000914, size=0x10, nextStart=0x4000980)
EMB(start=0x4000980, size=0x88, nextStart=0x0)
Now, release B.
dPartition(start=0x400090c, size=0x100, firstFreeStart=0x4000914)
EMB(start=0x4000914, size=0x34, nextStart=0x4000980)
EMB(start=0x4000980, size=0x88, nextStart=0x0)
At last, release C.
dPartition(start=0x400090c, size=0x100, firstFreeStart=0x4000914)
EMB(start=0x4000914, size=0xf4, nextStart=0x0)
Student >:_
```

> 在malloc申请的地址处成功init新的句柄进行子内存测试 申请0x100
>
> 分配过程
>
> EMB随着分配 size 0xf4(0x100-4)->0xe0(0xf4-0x10-4)->0xbc(0xe0-0x20-4)->0x88(0xbc-0x30-4)
>
> addr 0x14-(+0x10+4)>0x28-(+0x20+4)>0x4c-(+0x30+4)>0x80
>
> 正常
>
> 释放过程
>
> 从前往后
>
> 同样观察size与addr都正确
>
> 且释放第二块后仍然只有2块 合并成功
>
> 释放第三块后 只有一块 qie size addr 不变
>
> 正常

- testdP3

```
It is initialized as a very small dPartition,
dPartition(start=0x400090c, size=0x100, firstFreeStart=0x4000914)
EMB(start=0x4000914, size=0xf4, nextStart=0x0)
Now, A:B:C:- ==> A:B:-:- ==> A:- ==> - .
Alloc memBlock A with size 0x10: success(addr=0x4000918)!
dPartition(start=0x400090c, size=0xec, firstFreeStart=0x4000928)
EMB(start=0x4000928, size=0xe0, nextStart=0x0)
Alloc memBlock B with size 0x20: success(addr=0x400092c)!
dPartition(start=0x400090c, size=0xc8, firstFreeStart=0x400094c)
EMB(start=0x400094c, size=0xbc, nextStart=0x0)
Alloc memBlock C with size 0x30: success(addr=0x4000950)!
dPartition(start=0x400090c, size=0x94, firstFreeStart=0x4000980)
EMB(start=0x4000980, size=0x88, nextStart=0x0)
At last, release C.
dPartition(start=0x400090c, size=0xc8, firstFreeStart=0x400094c)
EMB(start=0x400094c, size=0xbc, nextStart=0x0)
Now, release B.
dPartition(start=0x400090c, size=0xec, firstFreeStart=0x4000928)
EMB(start=0x4000928, size=0xe0, nextStart=0x0)
Now, release A.
dPartition(start=0x400090c, size=0x100, firstFreeStart=0x4000914)
EMB(start=0x4000914, size=0xf4, nextStart=0x0)
```

> 分配过程和释放过程
>
> addr 和size 对称 所以释放过程正常
>
> 看分配过程
>
> EMB随着分配 size 0xf4 (0x100-4)->0xe0 (0xf4-0x10-4)->0xbc(0xe0-0x20-4)->0x88 (0xbc-0x30-4)
>
> addr 0x14--(+0x10+4)->0x28--(+0x20+4)->0x4c--(+0x30+4)->0x80
>
> 正常

- testeFP

```
EEB(start=0x4000984, next=0x0)
Alloc memBlock D, start = 0x4000988: 0xdddddddd
eFPartition(start=0x400090c, totalN=0x9c, perSize=0x20, firstFree=0x0)
Alloc memBlock E, failed!
eFPartition(start=0x400090c, totalN=0x9c, perSize=0x20, firstFree=0x0)
Now, release A.
eFPartition(start=0x400090c, totalN=0x9c, perSize=0x20, firstFree=0x4000918)
EEB(start=0x4000918, next=0x0)
Now, release B.
eFPartition(start=0x400090c, totalN=0x9c, perSize=0x20, firstFree=0x400093c)
EEB(start=0x400093c, next=0x4000918)
EEB(start=0x4000918, next=0x0)
Now, release C.
eFPartition(start=0x400090c, totalN=0x9c, perSize=0x20, firstFree=0x4000960)
EEB(start=0x4000960, next=0x400093c)
EEB(start=0x400093c, next=0x4000918)
EEB(start=0x4000918, next=0x0)
Now, release D.
eFPartition(start=0x400090c, totalN=0x9c, perSize=0x20, firstFree=0x4000984)
EEB(start=0x4000984, next=0x4000960)
EEB(start=0x4000960, next=0x400093c)
EEB(start=0x400093c, next=0x4000918)
EEB(start=0x4000918, next=0x0)
Student >:_
```

> 释放后的块都等距离 0x24
>
> 分配和释放正常

测试完后

```
maxMallocSizeNow
MAX_MALLOC_SIZE: 0x4001000 (with step = 0x1000);
```

maxsize正常

内核测试

- ktestmalloc

```
Student >:maxkallocSizeNow
MAX_MALLOC_SIZE: 0x3efa000 (with step = 0x1000);
Student >:ktestMalloc1
We allocated 2 buffers.
BUF1(size=19, addr=0x106afc) filled with 17(*): *****************
BUF2(size=24, addr=0x106b14) filled with 22(#): ######################

Student >:ktestMalloc2
We allocated 2 buffers.
BUF1(size=9, addr=0x106afc) filled with 9(+): +++++++++
BUF2(size=19, addr=0x106b0c) filled with 19(,): ,,,,,,,,,,,,,,,,,,,

Student >:maxkallocSizeNow
MAX_MALLOC_SIZE: 0x3efa000 (with step = 0x1000);
```

成功输出且 前后ksize一致

- ktestdP1

```
Student >:ktestdP1
We had successfully kalloc() a small memBlock (size=0x100, addr=0x106afc);
It is initialized as a very small dPartition;
dPartition(start=0x106afc, size=0x100, firstFreeStart=0x106b04)
EMB(start=0x106b04, size=0xf4, nextStart=0x0)
Alloc a memBlock with size 0x10, success(addr=0x106b08)!......Relaesed;
Alloc a memBlock with size 0x20, success(addr=0x106b08)!......Relaesed;
Alloc a memBlock with size 0x40, success(addr=0x106b08)!......Relaesed;
Alloc a memBlock with size 0x80, success(addr=0x106b08)!......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=0x106b08)!......Relaesed;
Alloc a memBlock with size 0x40, success(addr=0x106b08)!......Relaesed;
Alloc a memBlock with size 0x20, success(addr=0x106b08)!......Relaesed;
Alloc a memBlock with size 0x10, success(addr=0x106b08)!......Relaesed;
```

地址一致，0x100未申请成功

正常

- ktestdP2

```
dPartition(start=0x106afc, size=0x100, firstFreeStart=0x106b04)
EMB(start=0x106b04, size=0xf4, nextStart=0x0)
Now, A:B:C:- ==> -:B:C:- ==> -:C- ==> - .
Alloc memBlock A with size 0x10: success(addr=0x106b08)!
dPartition(start=0x106afc, size=0xec, firstFreeStart=0x106b18)
EMB(start=0x106b18, size=0xe0, nextStart=0x0)
Alloc memBlock B with size 0x20: success(addr=0x106b1c)!
dPartition(start=0x106afc, size=0xc8, firstFreeStart=0x106b3c)
EMB(start=0x106b3c, size=0xbc, nextStart=0x0)
Alloc memBlock C with size 0x30: success(addr=0x106b40)!
dPartition(start=0x106afc, size=0x94, firstFreeStart=0x106b70)
EMB(start=0x106b70, size=0x88, nextStart=0x0)
Now, release A.
dPartition(start=0x106afc, size=0x100, firstFreeStart=0x106b04)
EMB(start=0x106b04, size=0x10, nextStart=0x106b70)
EMB(start=0x106b70, size=0x88, nextStart=0x0)
Now, release B.
dPartition(start=0x106afc, size=0x100, firstFreeStart=0x106b04)
EMB(start=0x106b04, size=0x34, nextStart=0x106b70)
EMB(start=0x106b70, size=0x88, nextStart=0x0)
At last, release C.
dPartition(start=0x106afc, size=0x100, firstFreeStart=0x106b04)
EMB(start=0x106b04, size=0xf4, nextStart=0x0)
```

> EMB随着分配 size 0xf4 (0x100-4)->0xe0 (0xf4-0x10-4)->0xbc(0xe0-0x20-4)->0x88 (0xbc-0x30-4)
>
> addr 0x4-(+0x10+4)>0x18-(+0x20+4)>0x3c-(+0x30+4)>0x70
>
> 正常
>
> 释放过程合并操作正确，start正常

- ktestdP3

```
Machine   View
e had successfully kalloc() a small memBlock (size=0x100, addr=0x106afc);
t is initialized as a very small dPartition;
Partition(start=0x106afc, size=0x100, firstFreeStart=0x106b04)
MB(start=0x106b04, size=0xf4, nextStart=0x0)
ow, A:B:C:- ==> -:B:C:- ==> -:C- ==> - .
lloc memBlock A with size 0x10: success(addr=0x106b08)!
Partition(start=0x106afc, size=0xec, firstFreeStart=0x106b18)
MB(start=0x106b18, size=0xe0, nextStart=0x0)
lloc memBlock B with size 0x20: success(addr=0x106b1c)!
Partition(start=0x106afc, size=0xc8, firstFreeStart=0x106b3c)
MB(start=0x106b3c, size=0xbc, nextStart=0x0)
lloc memBlock C with size 0x30: success(addr=0x106b40)!
Partition(start=0x106afc, size=0x94, firstFreeStart=0x106b70)
MB(start=0x106b70, size=0x88, nextStart=0x0)
t last, release C.
Partition(start=0x106afc, size=0xc8, firstFreeStart=0x106b3c)
MB(start=0x106b3c, size=0xbc, nextStart=0x0)
ow, release B.
Partition(start=0x106afc, size=0xec, firstFreeStart=0x106b18)
MB(start=0x106b18, size=0xe0, nextStart=0x0)
ow, release A.
Partition(start=0x106afc, size=0x100, firstFreeStart=0x106b04)
MB(start=0x106b04, size=0xf4, nextStart=0x0)
tudent >:
                                                                19:33:01
```

前后一致 分配释放正常

- ktesteFP

```
EEB(start=0x106b74, next=0x0)
Alloc memBlock D, start = 0x106b78: 0xdddddddd
eFPartition(start=0x106afc, totalN=0x9c, perSize=0x20, firstFree=0x0)
Alloc memBlock E, failed!
eFPartition(start=0x106afc, totalN=0x9c, perSize=0x20, firstFree=0x0)
Now, release A.
eFPartition(start=0x106afc, totalN=0x9c, perSize=0x20, firstFree=0x106b08)
EEB(start=0x106b08, next=0x0)
Now, release B.
eFPartition(start=0x106afc, totalN=0x9c, perSize=0x20, firstFree=0x106b2c)
EEB(start=0x106b2c, next=0x106b08)
EEB(start=0x106b08, next=0x0)
Now, release C.
eFPartition(start=0x106afc, totalN=0x9c, perSize=0x20, firstFree=0x106b50)
EEB(start=0x106b50, next=0x106b2c)
EEB(start=0x106b2c, next=0x106b08)
EEB(start=0x106b08, next=0x0)
Now, release D.
eFPartition(start=0x106afc, totalN=0x9c, perSize=0x20, firstFree=0x106b74)
EEB(start=0x106b74, next=0x106b50)
EEB(start=0x106b50, next=0x106b2c)
EEB(start=0x106b2c, next=0x106b08)
EEB(start=0x106b08, next=0x0)
Student >:_
                                                                19:33:55
```

> 释放后的块都等距离 0x24
>
> 分配和释放正常

实验结果总结

测试用例中分配和释放均成功操作， 内核和用户分配的地址空间都在相应的地址上，该代码实现内存的分配与释放功能，且实现了内核与用户的分离