操作系统实验报告

实验四　Memory Management

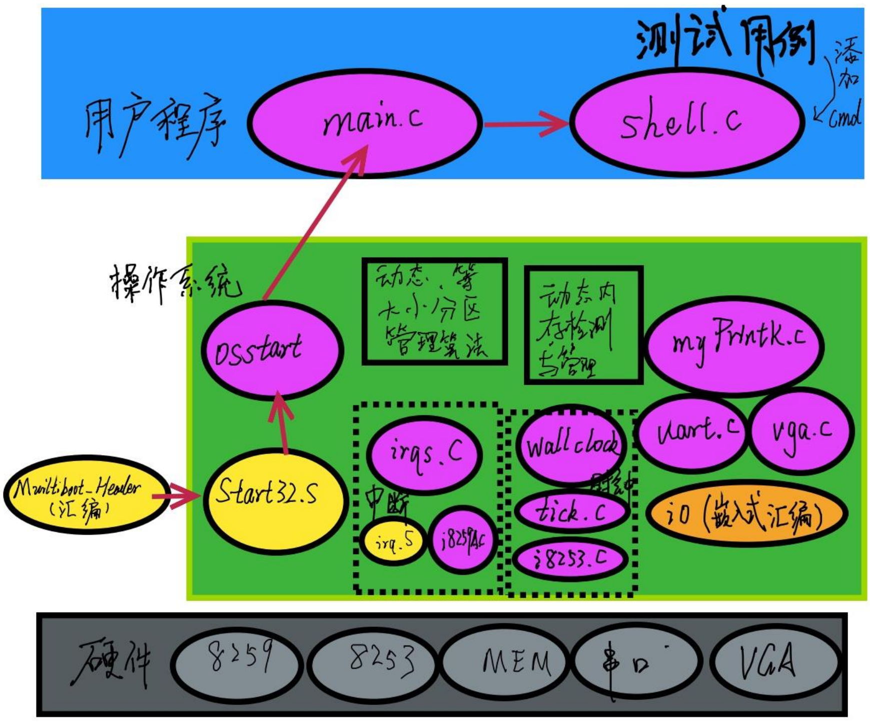
学号：PB18111683

姓名：童俊雄

完成时间：2020-05-09

一、　软件框图

紫色为C程序，黄色为汇编程序，橙色为嵌入式汇编

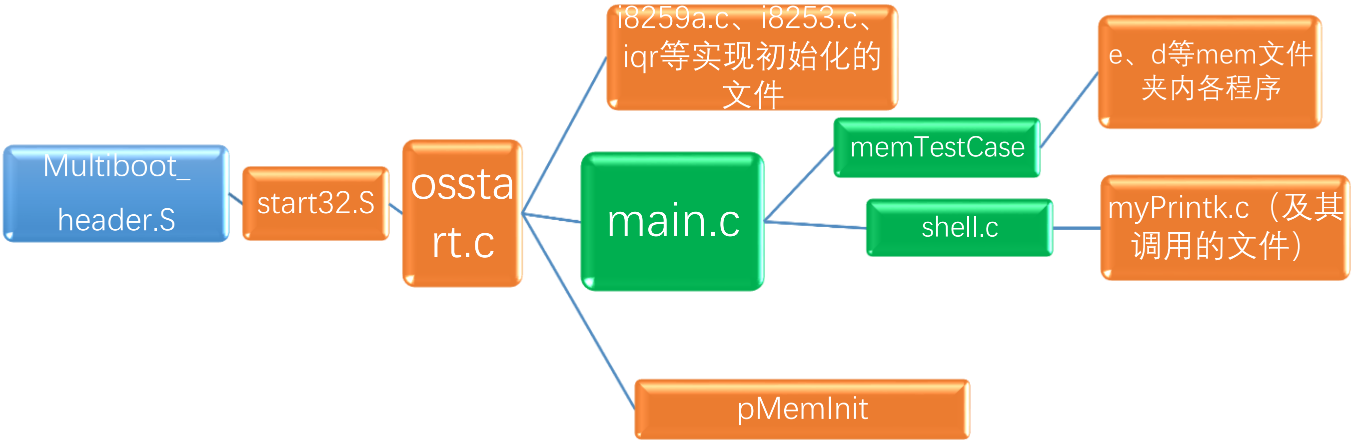


概述：软件大体上可以分为两个层次：用户程序和操作系统，其中操作系统又可以分为与用户程序的接口、IO设备的驱动程序、中断控制程序、时钟功能程序和内存管理程序等各功能模块，在每块中又可以进一步划分更细的层次（越上方的层次越高），如图所示。

二、　主流程说明

1. qemu启动header；
2. header通过调用myOS提供的_start，跳转到汇编文件start32.S；
   （进入myOS）
3. 从start32调用C语言入口，进入到c程序osstart.c；
4. osstart.c调用i8259A和i8253函数进行初始化，调用enable_interrupt函数启用中断；调用pMemInit接口进行内存检测；并调用用户程序的接口，即myMain函数，从而执行用户程序main.c；
   （进入UserApp）
5. main.c调用memtestinit接口以及shell接口；
6. memtest添加新的指令，并链接各指令对应的内存分配/管理函数
7. shell使用myprintk.c中的函数实现交互；

流程图：
（橙色表示myOS内的程序，绿色表示用户程序）



三、 主要功能模块说明&源代码说明

1. 内存检测模块

内存检测的主要函数：

Memtest：从start开始，以grainSize为步长，进行内存检测。由于start是无符号长整型（4byte），此处使用的检测方法为把start作为int型的指针，检测头尾的4个字节。

```c
void memTest(unsigned long start, unsigned long grainSize){
    /*本函数需要实现！！！*/
    /* ... */

    //最后，输出可用的内存的起始地址和大小，别忘记赋值给上面的全局变量
    if (start < 0x100000) {
        myPrintk(0x7, "the start location should be no less than 1M\n");
        return;
    }
    if (grainSize < 0x400) {
        myPrintk(0x7, "the grainsize should be no less than 1K\n");
        return;
    }
```

```c
*p = 0xaa55aa55;
if (*p != 0xaa55aa55)
    flag = 1;//结束标志
*p = 0x55aa55aa;
if (*p != 0x55aa55aa)
    flag = 1;
*p = i;
if (flag)break;
```

2. 等大小分区管理算法模块

结构体：

```c
// 一个EEB表示一个空闲可用的Block
struct EEB {
    unsigned long next_start;
};
```

```c
//eFPartition是表示整个内存的数据结构
struct eFPartition{
    unsigned long totalN;
    unsigned long perSize;  //unit: byte
    unsigned long firstFree;
};
```

这部分的主要函数有：

a) eFPartitionTotalSize，用于计算A的合理尺寸：

```c
unsigned long eFPartitionTotalSize(unsigned long perSize, unsigned long n){
    //本函数需要实现！！！
    /* ... */
    unsigned long persize;
    unsigned long all;
    //8字节对齐
    if ((perSize & 7) == 0)
        persize = perSize;
    else
        persize = perSize - (perSize & 7) + 8;

    all = persize * n + sizeof(struct eFPartition);
    return all;
}
```

b) eFPartitionInit，用于对A进行划分和管理：

```c
unsigned long eFPartitionInit(unsigned long start, unsigned long perSize, unsigned long n){
    //本函数需要实现！！！
    /* ... */
    unsigned long persize;
    if ((perSize & 7) == 0)
        persize = perSize;
    else
        persize = perSize - (perSize & 7) + 8;
    struct eFPartition* efp = (struct eFPartition*) start;
    efp->perSize = persize;
    efp->totalN = n;
    efp->firstFree = start + sizeof(struct eFPartition);
```

c) eFPartitionAlloc和eFPartitionFree，用于按需求分配和释放：

```c
unsigned long eFPartitionAlloc(unsigned long EFPHandler){
    //本函数需要实现！！！
    /*本函数分配一个空闲块的内存并返回相应的地址，EFPHandler表示整个内存的首地址
    */
    struct eFPartition* efp = (struct eFPartition*) EFPHandler;
    struct EEB* eeb = (struct EEB*) efp->firstFree;
    if (efp->firstFree > EFPHandler + eFPartitionTotalSize(efp->perSize, efp->totalN)) //failed
        return 0;

    efp->firstFree = eeb->next_start;//success
    return (unsigned long)eeb;
}

unsigned long eFPartitionFree(unsigned long EFPHandler,unsigned long mbStart){
    //本函数需要实现！！！
    /* ... */
    struct eFPartition* efp = (struct eFPartition*) EFPHandler;
    if (mbStart == 0) mbStart = EFPHandler + eFPartitionTotalSize(efp->perSize, efp->totalN);
    efp->firstFree = EFPHandler + sizeof(struct eFPartition);
    struct EEB* eeb = (struct EEB*) efp->firstFree;
    int i = 0;
    while ((unsigned long)eeb < mbStart) {
        eeb->next_start = (unsigned long)eeb + efp->perSize;
        eeb = (struct EEB*) ((unsigned long)eeb + efp->perSize);
        i++;
    }
    eeb = (struct EEB*) ((unsigned long)eeb - efp->perSize);
    if (i == efp->totalN)
        eeb->next_start = 0;
    return 1;
}
```

3. 动态分区管理算法模块

结构体：

```c
//dPartition 是整个动态分区内存的数据结构
struct dPartition{
    unsigned long size;
    unsigned long firstFreeStart;
};
```

```c
// EMB每一个block的数据结构，userdata可以暂时不用管。
struct EMB{
    unsigned long size;
    union {
        unsigned long nextStart;    // if free: pointer to next block
        unsigned long userData;     // if allocated, blongs to user
    };
};
```

a) dPartitionInit：对B进行初始化

```c
unsigned long dPartitionInit(unsigned long start, unsigned long totalSize){
    //本函数需要实现！！！
    /* ... */
    if (totalSize < sizeof(struct EMB) + sizeof(struct dPartition) )
        return 0;

    struct dPartition* dp = (struct dPartition*) start;
    dp->size = totalSize;
    dp->firstFreeStart = start + 8;

    struct EMB* emb = (struct EMB*)(dp->firstFreeStart);
    emb->size = totalSize - 16;
    emb->nextStart = 0;
    return start;
```

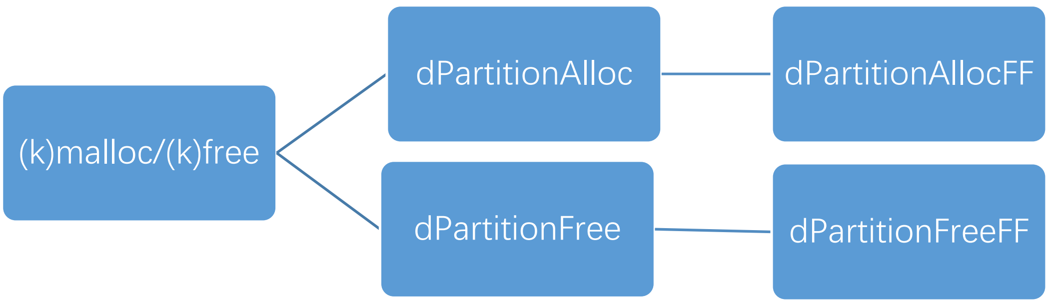
b) dPartitionAlloc和dPartitionFree：按需求分配和回收

```c
unsigned long dPartitionAlloc(unsigned long dp, unsigned long size){
    return dPartitionAllocFirstFit(dp, size);
}

unsigned long dPartitionFree(unsigned long   dp, unsigned long start){
    return dPartitionFreeFirstFit(dp,start);
}
```

调用的两个F-F算法函数功能如下：（实现代码较长，此处略）

```c
unsigned long dPartitionAllocFirstFit(unsigned long dp, unsigned long size){
    //本函数需要实现！！！
    /*
    使用firstfit的算法分配空间，当然也可以使用其他fit，不限制。
    最后，成功分配返回首地址，不成功返回0

unsigned long dPartitionFreeFirstFit(unsigned long dp, unsigned long start){
    //本函数需要实现！！！
    /*按照对应的fit的算法释放空间
    注意检查要释放的start~end这个范围是否在dp有效分配范围内
    返回1 没问题
    返回0 error
```

流程图：



4. shell模块:新增如下功能

```c
void addNewCmd( unsigned char *cmd,
        int (*func)(int argc, unsigned char **argv),
        void (*help_func)(void),
        unsigned char* description){
    // ...
    struct cmd* tcmd = (struct cmd*) malloc(sizeof(struct cmd));
    strcpy(cmd, tcmd->cmd);
    tcmd->func = func;
    tcmd->help_func = help_func;
    strcpy(description, tcmd->description);
    tcmd->nextCmd = NULL;
    struct cmd* tmpCmd = ourCmds;
    if (tmpCmd == NULL)
    {
        ourCmds = tcmd;
        return;
    }
    while (tmpCmd->nextCmd != NULL) {
        tmpCmd = tmpCmd->nextCmd;
    }
    tmpCmd->nextCmd = tcmd;
```

四、 目录组织

所有文件：

```
├── Makefile
├── README_mem.txt
├── multibootheader
│   └── multibootHeader.S
├── myOS
│   ├── Makefile
│   ├── dev
│   │   ├── Makefile
│   │   ├── i8253.c
│   │   ├── i8259A.c
│   │   ├── uart.c
│   │   └── vga.c
│   ├── i386
│   │   ├── Makefile
│   │   ├── io.c
│   │   ├── irq.S
│   │   └── irqs.c
│   ├── include
│   │   ├── i8253.h
│   │   ├── i8259.h
│   │   ├── i8259A.h
│   │   ├── io.h
│   │   ├── irq.h
│   │   ├── irqs.h
│   │   ├── kmalloc.h
│   │   ├── kmem.h
│   │   ├── malloc.h
│   │   ├── mem.h
│   │   ├── myPrintf.h
│   │   ├── myPrintk.h
│   │   ├── string.h
│   │   ├── tick.h
│   │   ├── uart.h
│   │   ├── vga.h
│   │   ├── vsprintf.h
│   │   └── wallClock.h
│   ├── kernel
│   │   ├── Makefile
│   │   ├── mem
│   │   │   ├── Makefile
│   │   │   ├── dPartition.c
│   │   │   ├── eFPartition.c
│   │   │   ├── malloc.c
│   │   │   └── pMemInit.c
│   │   ├── tick.c
│   │   └── wallClock.c
│   ├── lib
│   │   ├── Makefile
│   │   └── string.c
│   ├── myOS.ld
│   ├── osStart.c
│   ├── printk
│   │   ├── Makefile
│   │   ├── myPrintk.c
│   │   ├── types.h
│   │   └── vsprintf.c
│   ├── start32.S
│   └── userInterface.h
├── source2img.sh
└── userApp
    ├── Makefile
    ├── main.c
    ├── memTestCase.c
    ├── memTestCase.h
    ├── shell.c
    └── shell.h
```

Makefile组织：

见上图中的各Makefile文件

五、 代码布局

由myOS.ld的代码可知，myOS.elf文件中有三个 section：

1. 第一个section为.text，位置从1M处开始，在.text内的分布为8字节对齐，前12字节为魔术，从第16字节开始是代码部分；

代码结束后16位对齐；

2. 第二个section为.data，位置从.text结束并对齐后开始；
   末尾16位对齐；
3. 第三个section为.bss，位置从.data结未尾对齐后开始；
   末尾16位对齐；
4. .bss结束后是_end，此处是我们可以操作的内存空间的开始，512位对齐；

六、 编译过程说明

由makefile可知，编译过程有以下两步：

1. 编译汇编代码（header.S和start32.S）和C代码（osstart.c等）生成.o文件；
2. 根据myOS.ld的部署要求，把上述.o文件链接成myOS.elf文件
   如下图所示：

```
ld -n -T myOS/myOS.ld output/multibootheader/multibootHeader.o output/myOS/start32.o outpu
/myOS/osStart.o output/myOS/dev/uart.o output/myOS/dev/vga.o output/myOS/dev/i8253.o outpu
/myOS/dev/i8259A.o output/myOS/i386/io.o output/myOS/i386/irq.o output/myOS/i386/irqs.o ou
put/myOS/printk/myPrintk.o output/myOS/lib/string.o output/myOS/kernel/tick.o output/myOS/
ernel/wallClock.o output/myOS/kernel/mem/pMemInit.o output/myOS/kernel/mem/dPartition.o ou
put/myOS/kernel/mem/eFPartition.o output/myOS/kernel/mem/malloc.o output/userApp/main.o ou
put/userApp/shell.o output/userApp/memTestCase.o -o output/myOS.elf
make succeed
```

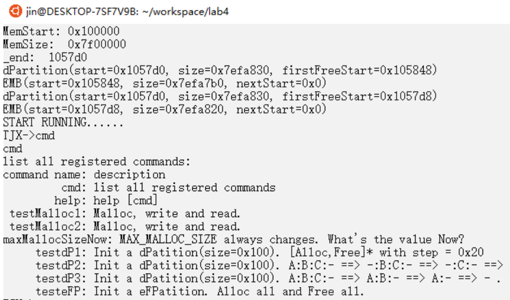
七、 运行和运行结果说明

运行指令qemu-system-i386 -kernel output/myOS.elf -serial stdio

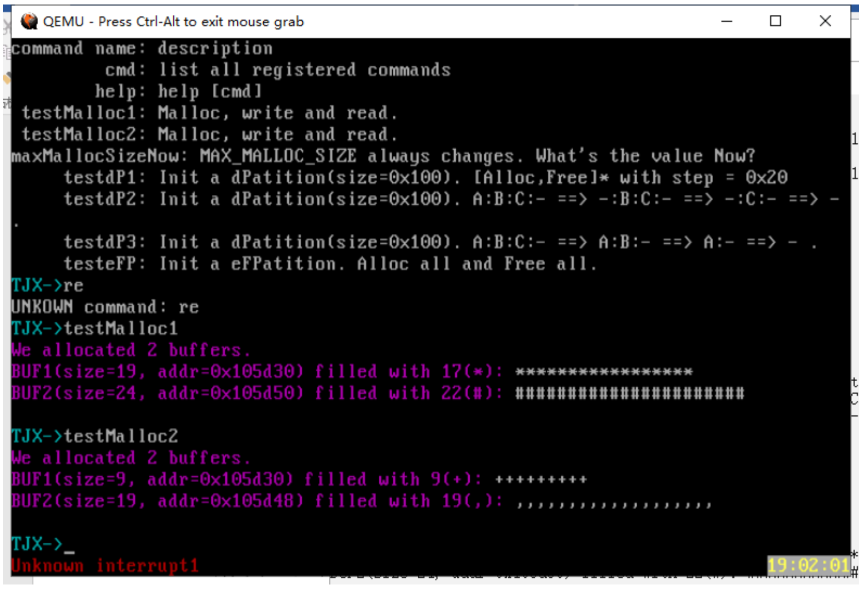
运行指令sudo screen /dev/pts/0

运行结果如下如所示：

1. 内存检测



2. 功能函数：



```
TJX->testdP1
We had successfully malloc() a small memBlock (size=0x100, addr=0x105f50);
It is initialized as a very small dPartition;
dPartition(start=0x105f50, size=0x100, firstFreeStart=0x105f58)
EMB(start=0x105f58, size=0xf0, nextStart=0x0)
Alloc a memBlock with size 0x10, success(addr=0x105f58)!......Relaesed;
Alloc a memBlock with size 0x20, success(addr=0x105f58)!......Relaesed;
Alloc a memBlock with size 0x40, success(addr=0x105f58)!......Relaesed;
Alloc a memBlock with size 0x80, success(addr=0x105f58)!......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=0x105f58)!......Relaesed;
Alloc a memBlock with size 0x40, success(addr=0x105f58)!......Relaesed;
Alloc a memBlock with size 0x20, success(addr=0x105f58)!......Relaesed;
Alloc a memBlock with size 0x10, success(addr=0x105f58)!......Relaesed;
TJX->
```

jin@DESKTOP-7SF7V9B: ~/workspace/lab4

```
Alloc a memBlock with size 0x10, success(addr=0x105d38)!......Relaesed;
TJX->testdP2
testdP2
We had successfully malloc() a small memBlock (size=0x100, addr=0x105e40);
It is initialized as a very small dPartition;
dPartition(start=0x105e40, size=0x100, firstFreeStart=0x105e48)
EMB(start=0x105e48, size=0xf0, nextStart=0x0)
Now, A:B:C:- ==> -:B:C:- ==> -:C- ==> - .
Alloc memBlock A with size 0x10: success(addr=0x105e48)!
dPartition(start=0x105e40, size=0x100, firstFreeStart=0x105e68)
EMB(start=0x105e68, size=0xd0, nextStart=0x0)
Alloc memBlock B with size 0x20: success(addr=0x105e68)!
dPartition(start=0x105e40, size=0x100, firstFreeStart=0x105e98)
EMB(start=0x105e98, size=0xa0, nextStart=0x0)
Alloc memBlock C with size 0x30: success(addr=0x105e98)!
dPartition(start=0x105e40, size=0x100, firstFreeStart=0x105ed8)
EMB(start=0x105ed8, size=0x60, nextStart=0x0)
Now, release A.
```

jin@DESKTOP-7SF7V9B: ~/workspace/lab4

```
Alloc a memBlock with size 0x10, success(addr=0x105f58)!......Relaesed;
TJX->testdP3
testdP3
We had successfully malloc() a small memBlock (size=0x100, addr=0x106060)
It is initialized as a very small dPartition;
dPartition(start=0x106060, size=0x100, firstFreeStart=0x106068)
EMB(start=0x106068, size=0xf0, nextStart=0x0)
Now, A:B:C:- ==> A:B:- ==> A:- ==> - .
Alloc memBlock A with size 0x10: success(addr=0x106068)!
dPartition(start=0x106060, size=0x100, firstFreeStart=0x106088)
EMB(start=0x106088, size=0xd0, nextStart=0x0)
Alloc memBlock B with size 0x20: success(addr=0x106088)!
dPartition(start=0x106060, size=0x100, firstFreeStart=0x1060b8)
EMB(start=0x1060b8, size=0xa0, nextStart=0x0)
Alloc memBlock C with size 0x30: success(addr=0x1060b8)!
dPartition(start=0x106060, size=0x100, firstFreeStart=0x1060f8)
EMB(start=0x1060f8, size=0x60, nextStart=0x0)
Now, release C.
```

jin@DESKTOP-7SF7V9B: ~/workspace/lab4

```
EEB(start=0x105d99, next=0x0)
Alloc memBlock C, start = 0x105d7a: 0xcccccccc
eFPartition(start=0x105d30, totalN=0x4, perSize=0x20, firstFree=0x105d99)
EEB(start=0x105d99, next=0x0)
Alloc memBlock D, start = 0x105d99: 0xdddddddd
eFPartition(start=0x105d30, totalN=0x4, perSize=0x20, firstFree=0x0)
Alloc memBlock E, failed!
eFPartition(start=0x105d30, totalN=0x4, perSize=0x20, firstFree=0xf000ff53)
EEB(start=0xf000ff53, next=0x0)
Now, release A.
eFPartition(start=0x105d30, totalN=0x4, perSize=0x20, firstFree=0x105d3c)
EEB(start=0x105d3c, next=0xaaaaaaaa)
EEB(start=0xaaaaaaaa, next=0x0)
Now, release B.
eFPartition(start=0x105d30, totalN=0x4, perSize=0x20, firstFree=0x105d3c)
EEB(start=0x105d3c, next=0x105d5c)
EEB(start=0x105d5c, next=0xbbbbbbb)
EEB(start=0xbbbbbbb, next=0x0)
Now, release C.
eFPartition(start=0x105d30, totalN=0x4, perSize=0x20, firstFree=0x105d3c)
EEB(start=0x105d3c, next=0x105d5c)
EEB(start=0x105d5c, next=0x105d7c)
EEB(start=0x105d7c, next=0xcccc)
EEB(start=0xcccc, next=0x0)
Now, release D.
```

八、 遇到的问题和解决办法

问题：mentest使用short指针读写失败；

解决：询问了朱同学，利用int指针解决了这一问题。