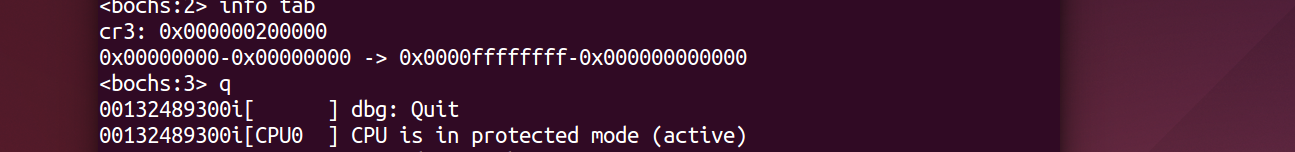
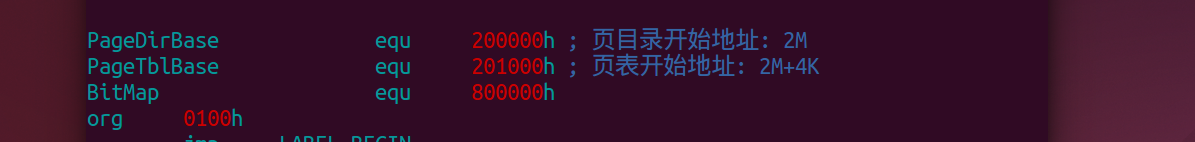
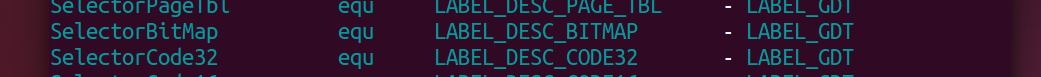
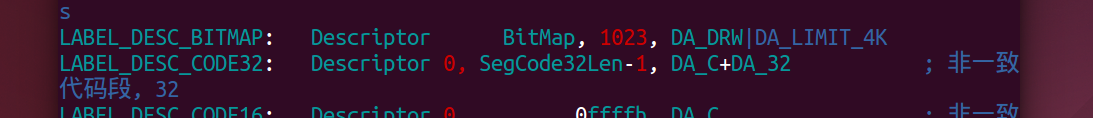
由下图可知，我们共有0x100000000字节的物理空间，也就是0x10000个物理页，所以我们共需要0x100000个bit即0x20000个字节作为bitmap；

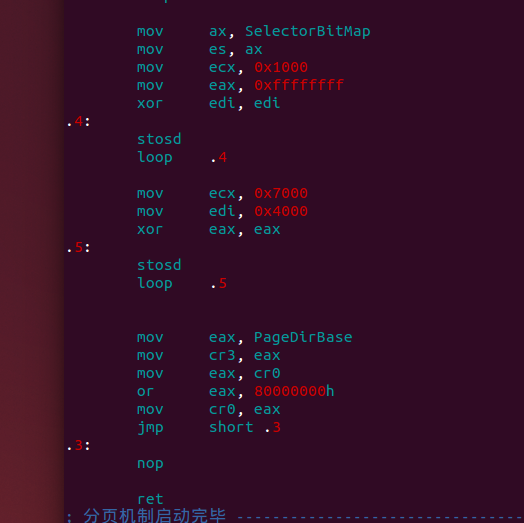


开辟bitmap的空间，仿照页表的空间开辟，为bitmap专门注册一个段；





对bitmap进行初始化，规定前0x8000个物理页框已经被占用，后边的为空闲：



ALLOC部分代码如下：

[SECTION .s32]

[BITS   32]

LABEL\_ALLOC:

;edi --> start address

;esi --> size

    push    edx

    push    ecx

    push    ebx

;采用的是最先适应算法，其中edi寄存器存放待分配空间所希望使用的线性地址，esi存放空间的大小、

;使用一个叫作bitmap的数据结构，每一位表示一个物理页是否空闲，0表示空闲，1表示忙碌

    shr esi, 12 ;计算需要多少物理页框

    mov edx, esi

    shr edx, 3  ;将页框数量除以8，得到bitmap中的字节数

    inc edx

    mov ax, SelectorBitMap

    mov es, ax

    mov ecx, edx

    push    edi

    xor edi, edi

;下面这个循环用来寻找连续的符合size大小要求的物理内存，其中为了加快效率，我们采用一个字节比对，即所计算的空闲物理页框为8的倍数，但实际分配仍然是刚好的；

.1:

    mov al, [es:edi]

    cmp al, 0

jz  .2

;如果中间断开了，则要重新开始

    mov ecx, edx

    inc ecx ;下面有个loop会导致ecx--

.2:

    inc edi

    loop    .1

sub edi, edx ; 找到之后还要将edi回退到这篇空闲区域的起始编号

    mov ecx, edx

    mov eax, edi

    mov bl, 0xff

;将相关的bitmap位改为忙碌

.3:

    mov [es:eax], bl

    inc eax

    loop    .3

    mov eax, edi

    pop edi ; start address

    push    eax

;修改对应页表项

    mov ax, SelectorPageTbl

    mov es, ax

    shr edi, 12 ; liner address --> liner page index

    shl edi, 2  ; PTE offset

    mov ecx, esi ; the number of pages

    pop eax   ; the start bitmap **int**

    shl eax, 3 ; the start pysical page

    shl eax, 12 ; the start physical address

    add     eax, PG\_P  | PG\_USU | PG\_RWW

    shr ecx, 1

.4:

    stosd

    add eax, 4096

    add edi, 4

    loop    .4

    pop ebx

    pop ecx

    pop edx

    retf

ALLOC\_LEN   equ $ - LABEL\_ALLOC

FREE部分代码如下：

[SECTION .s32]

[BITS   32]

LABEL\_FREE:

    ;function free

    ;rdi --> the start address

    ;rsi --> the size

    push    edx

    push    ecx

    push    ebx

    shr esi, 12 ; the number of pages

    mov edx, esi

    shr edx, 3  ; the bytes of bitmap

    mov     ax, SelectorPageTbl

        mov     es, ax

    mov ebx, edi

    shr ebx, 12 ; start liner address page index

    shl ebx, 2  ; start liner address PTE offset

    mov ecx, esi

    ;clean the page table

    xor eax, eax

.1:

    mov [es:ebx], al

    add ebx, 4

    loop    .1

    mov ax, SelectorBitMap

        mov     es, ax

    mov ecx, edx

    shr edi, 12 ; start liner address page index

    shr edi, 3  ; start liner address in bitmap bytes index

    xor eax, eax

.2:

    mov [es:edi], al

    loop    .2

    pop ebx

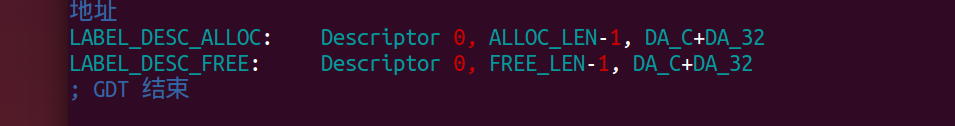
    pop ecx

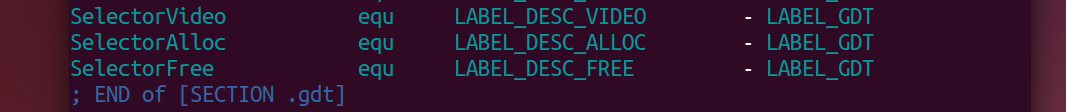
    pop ebx

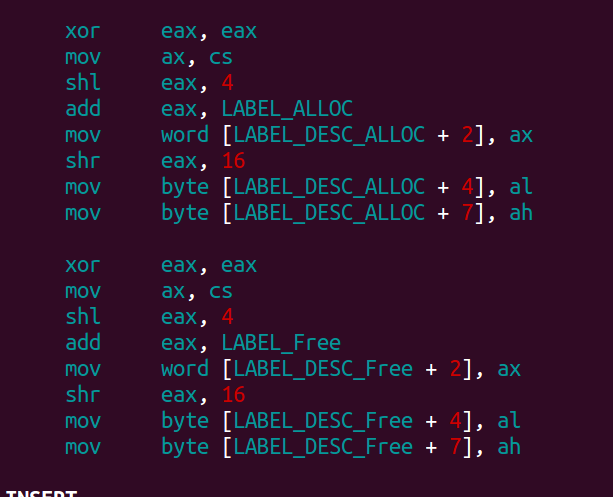
    retf

FREE\_LEN    equ $ - LABEL\_FREE

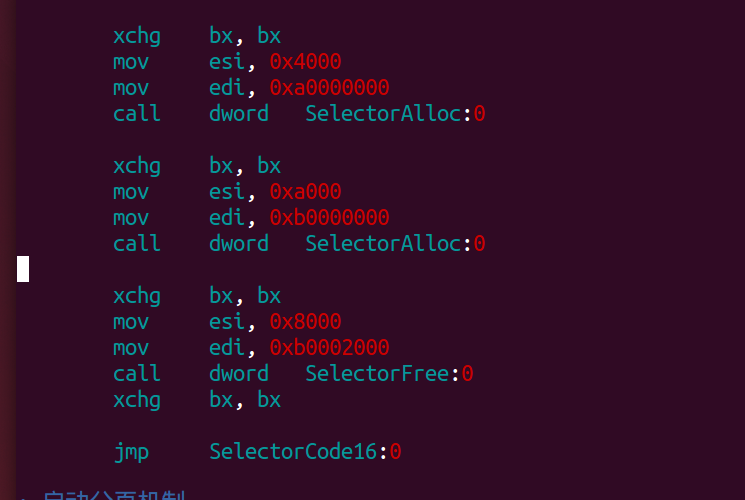
为它们添加段选择子、段描述符，并初始化：



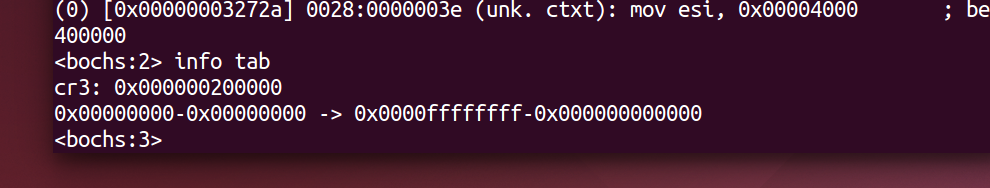




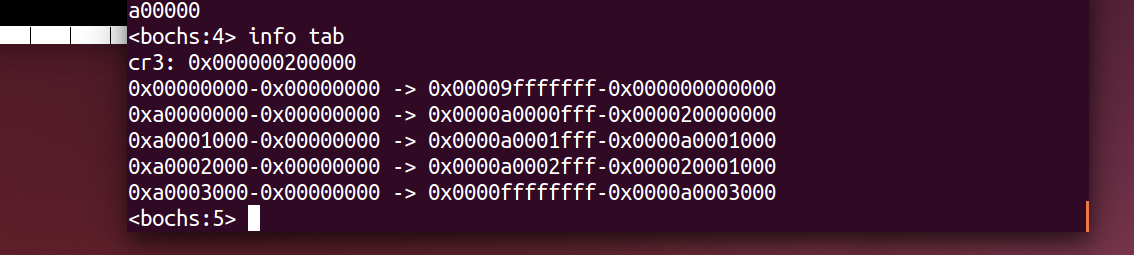
下面展示其效果，如下图所示，分别有三次函数调用，第一次申请0x4000字节内存，起始线性地址位0xA0000000；第二次申请0xA000字节内存，起始线性地址为0xB0000000；第三次释放掉起始线性地址为0xB0002000的0x8000字节内存；分别在调用前后下断点；



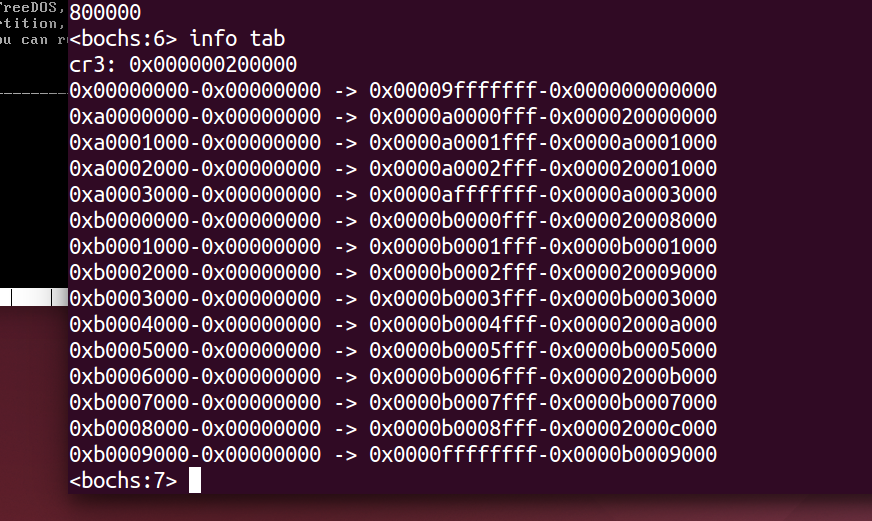
最开始的内存映射情况如下图所示：



第一次ALLOC后的内存映射情况如下图所示，可以看到分配了四个物理页框：



第二次ALLOC后的内存映射情况如下图所示，可以看到又分配了10个物理页框：



FREE之后的内存映射情况如下图所示，可以看到减少了相应的8个物理页框的映射：

