武汉大学国家网络安全学院教学实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| **课程名称** | **操作系统设计与实践** | **实验日期** | **2023.10.23** |
| **实验名称** | **分页机制** | **实验周次** | **第三周** |
| **姓名** | **学号** | **专业** | **班级** |
| 程子洋 | 2021301051114 | 网络空间安全 | 9班 |
| 聂森 | 2021302191536 | 网络空间安全 | 9班 |
| 王卓 | 2021302191791 | 网络空间安全 | 9班 |
| 刘虓 | 2021302121234 | 网络空间安全 | 9班 |

一、 实验目的及实验内容

（本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析）

**1** **.** **1** 实验知识要求

1. 深入理解分页机制的工作原理, 包括页表、页目录、 CR3 寄存器等的概念。

2. 理解分页机制启动的步骤, 包括页表的初始化和 CR0 寄存器的设置。

3. 理解段页机制是如何在保护模式下工作的, 以及为什么需要配合使用。

4. 掌握如何通过 BIOS 中断(如 int 15h)来获取系统内存信息, 特别是在实模式下的操作。

5. 了解页表项(Page Table Entry,PTE) 和页目录项(Page Directory Entry,PDE) 的结构和初始化 方式。

6. 理解如何计算物理地址从虚拟地址的转换过程, 包括如何查找页表和页目录表。

7. 掌握页表和页目录表的管理方法, 如页的分配和释放, 以及位图管理方式。

8. 理解 CPU 访问内存时如何使用 CR3 寄存器、页目录表和页表来进行地址转换。

9. 理解关于分页管理的进阶话题, 如页面置换算法(如果有相关内容)。

**1** **.** **2** 实验内容

**1** **.** **2** **.** **1** 本次实验内容

1. 认真阅读章节资料, 掌握什么是分页机制

2. 调试代码, 掌握分页机制基本方法与思路 代码 *3.22* 中*,212* 行 *—* *237* 行*,* 设置断点调试这几个循 环*,* 分析究竟在这里做了什么？

3. 掌握 PDE,PTE 的计算方法 (a) 动手画一画这个映射图

(b) 为什么代码 3.22 里面,PDE 初始化添加了一个 PageTblBase(Line 212), 而 PTE 初始化时候 没有类似的基地址呢(Line224)？

4. 熟悉如何获取当前系统内存布局的方法

5. 掌握内存地址映射关系的切换 画出流程图

6. 基础题：

(a) 依据实验的代码, 自定义一个函数, 给定一个虚拟地址, 能够返回该地址从虚拟地址到物理地

址的计算过程, 如果该地址不存在, 则返回一个错误提示。

(b) 完善分页管理功能, 补充 alloc\_pages, free\_pages 两个函数功能, 试试你能一次分配的最大

空间有多大, 如果超出了有什么办法解决呢？

7. 进阶题(选做)： 设计一个内存管理器, 提示, 均按照页为最小单位进行分配、对于空闲空间管理 可采用位图法或其他方法进行管理, 分配策略不限。

**1** **.** **2** **.** **2** 思考题

1. 分页和分段有何区别？在本次实验中, 段页机制是怎么搭配工作的？

2. PDE、PTE, 是什么？ 例程中如何进行初始化？ CPU 是怎样访问到 PDE、PTE, 从而计算出物理 地址的？

3. 开启分页机制之后, 在 GDT 表中、在 PDE 、PTE 中存的地址是物理地址、线性地址, 还是逻辑 地址, 为什么？

4. 为什么 PageTblBase 初始值为 2M+4K？能不能比这个值小？

5. 怎么读取本机的实际物理内存信息？

6. 如何进行地址映射与切换？

7. 如何实现 alloc\_pages , free\_pages

**1** **.** **3** 原理分析

1. 分页机制——启动分页的步骤:

(a) 初始化页目录表、页表、 CR3 的内容, 使各部分指向正确的位置。

(b) 修改 CR0 的 PG 位, 使分页机制打开。

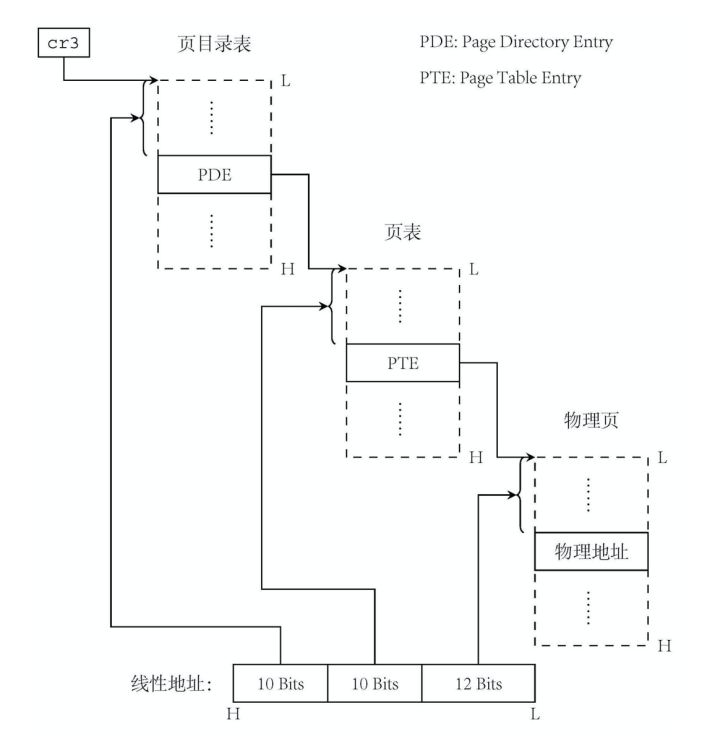


图 1: 分页机制

2. 取得当前物理内存信息:

在 BIOS 中断 int 15h 中, 循环读取 ARDS 结构(Address Range Descriptor Struct) 通常是在实 模式下进行的。这是因为实模式和保护模式是 x86 处理器的两种不同工作模式, 它们对内存访问 和地址空间有不同的规则和限制。实模式和保护模式下内存变量的访问问题如下：

(a) 实模式：

i. 实模式是 x86 处理器的启动默认模式, 它提供了较少的内存管理和保护机制。

ii. 在实模式下, 物理内存可寻址空间为 1MB, 使用 16 位寻址。访问内存通常是直接的, 没 有分段或分页保护。这

iii. 意味着程序可以自由地访问整个物理内存, 但也容易导致内存冲突和错误。

iv. BIOS 例程通常在实模式下运行, 因此在这种情况下, 可以直接访问物理内存。 (b) 保护模式：

i. 保护模式提供了更强大的内存管理和保护机制, 支持 32 位寻址, 以及分段和分页机制。

ii. 在保护模式下, 程序需要遵守严格的内存保护规则, 包括使用段描述符来限制对内存的访 问, 以及使用分页机制来将虚拟地址映射到物理地址。

iii. 保护模式下, 程序无法直接访问整个物理内存, 因为物理内存是受分页和段描述符的保护 的。

对于 BIOS 中断 int 15h, 由于它通常由 BIOS 例程提供, 在实模式下执行, 因此在这种情况下, 程序可以利用实模式的特性直接访问物理内存。ARDS 结构包含了系统的内存信息, 例如可用内存段的起始地址和长度, 因此在实模式下可以轻松地读取这些信息, 而不受保护模式的限制。

3. 计算物理页的个数:

(a) 从 ARDS 结构中获取系统内存信息。 ARDS 结构包含了描述可用内存段的信息, 其中包括每

个内存段的起始地址和长度。

(b) 将所有可用内存段的长度相加, 以获取总的可用内存大小。并获取正在使用的页面大小。

(c) 使用总的可用内存大小除以页面大小, 然后向上取整。

4. 实现内存页的分配:

(a) 初始化位图

i. 假设内存大小, 首先需要知道物理内存的大小, 以确定位图的大小和页面数量。

ii. 假设可用空间, 确定哪些部分的物理内存是可供分配的。

(b) 根据 cr3 寄存器和线性地址查找页目录表。

i. 若无, 则创建该表项。

ii. 若有, 则根据表项中的页表基址和线性地址查找页表。

(c) 根据页数量, 循环查询位图中的空闲内存地址, 并赋值给页表项作为页基址。

5. 实现内存页的释放:

(a) 标记页面为空闲：当需要释放一个内存页时, 首先需要将相应的页面标记为空闲。通常, 这涉

及到更新一个内存管理数据结构, 如位图, 以指示该页面现在是空闲的状态。

(b) 解除页表中的映射：在分页系统中, 每个内存页都与页表中的一个表项相关联。为了释放页

面, 需要将页表中的相关表项更新, 以取消该页面与虚拟地址的映射。

(c) 清空页面内容：为了安全起见, 通常建议在释放页面时将其内容清空。这可以防止敏感数据 泄漏, 因为该页面可能在稍后被重新分配给其他进程。

(d) 回收页表项：如果需要, 还可以回收与该页面相关的页表项, 以便在以后的内存分配中重用。

6. 相关汇编知识:

(a) Stosd (dword,4 字节)

i. 将 eax 的内容(4 字节) 存储到 es:edi 指向的内存单元中, 同时 edi 的值根据方向标志的 值 (DF==0) 增加或者减少(DF==1)。

ii. 相应的还有 stosb ( 1 字节) ,stosw (2 字节)。

(b) 如何在 bochs 里面对.com 文件进行调试

i. 使用 Magic break, 修改 bochsrc 文件, 在文件末尾添加”magic\_break: enabled=1 ”。

ii. 在需要添加调试的语句前面增加 xchg bx,bx, 则执行时候 bochs 就会停下来。调试时候 n 是下一句, 会跳过函数调用, 如果要进去, 需要 s (step in)进入那个函数。

二、 实验环境及实验步骤

（本次实验所使用的器件、仪器设备等的情况；具体的实验步骤）

**2** **.** **1** 实验环境

• 虚拟机： VMware Workstation Pro/VMware Workstation 16 player

• 操作系统： Ubuntu 16.04

• 模拟系统软件： Bochs 2.7

**2** **.** **2** 具体实验步骤

**2** **.** **2** **.** **1** 认真阅读章节资料**,** 掌握什么是分页机制

分页机制是在分段机制之后进行的， 分段机制将逻辑地址转换成线性地址， 分页机制进一步将线 性地址转换为物理地址。分页机制就像一个函数， 通过某种计算方式将线性地址映射到对应的物理地 址上。

**2** **.** **2** **.** **2** 调试代码**,** 掌握分页机制基本方法与思路

代码 3.22 中,212 行 —— 237 行，设置断点调试这几个循环，分析究竟在这里做了什么？为了实现调 试功能，我们在 bochsrc 中添加 magic\_break: enabled=1:

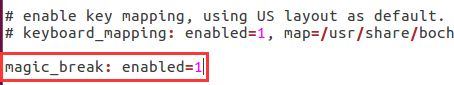


图 2: bochsrc 修改

在第 212 行代码之前设置断点进行调试：

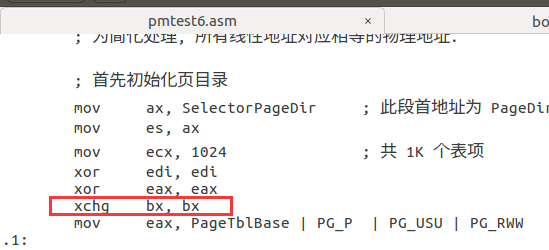


图 3: 设置断点

这里要分析循环是在干什么即分析寄存器值的变化(重点在 eax 寄存器的值)，接下来是调试过程：

1. 查看最初寄存器的值:

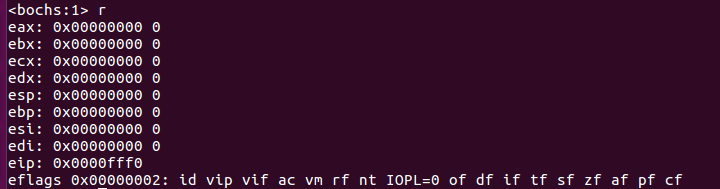


图 4: 最初寄存器的值

2. 输入 c ，使程序运行到断点，并使用命令 trace-reg on 使后续调试一直显示寄存器值的内容:

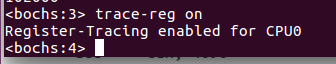


图 5: 命令 trace-reg on

3. 查看寄存器的值:

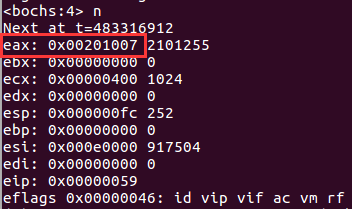


图 6: 寄存器的值

这里到运行到 mov eax, PageTblBase | PG\_P | PG\_USU | PG\_RWW。作用为将。 PageTblBase 以及属性的值赋给 eax ，则 eax = 201000h + 1 + 2 + 4 =201007h。

4. 继续运行到 loop .1 处，完成页目录表的初始化:

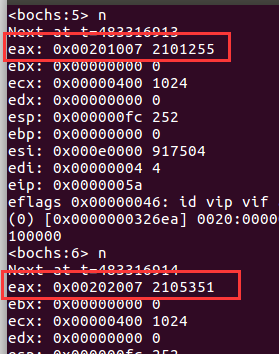


图 7: 继续查看寄存器的值 1

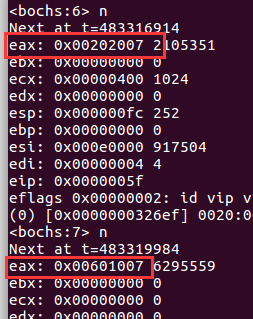


图 8: 继续查看寄存器的值 2

在 bochs:5 处运行到 stosd 处，作用是将 eax 中的 PageTblBase | PG\_P | PG\_USU | PG\_RWW 复制到 edi 的内存空间， 复制四个字节， 并将 edi 增加 4 个字节， 即将 eax 的内容存入第一个 PDE，让当前第一个 PDE 对应的页表首地址变成 PageTblBase。

在 bochs:6 处运行到 add eax，4096 处，作用是使所有页表在内存中是连续的， eax=PageTblBase 基址 +4096。

在 bochs:7 处运行到 loop .1 处， 这里经过 1024 次循环， eax=PageTblBase 的基址 +1024 个 4096，页目录表中的所有 PDE 都被复制完毕，页目录表的初始化完成。

5. 继续单步执行，完成页表的初始化:

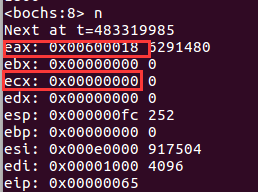


图 9: 页表初始化 1

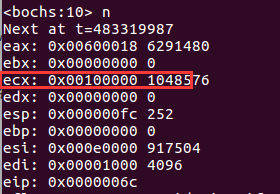


图 10: 页表初始化 2

在 bochs:8 处运行到 mov ax, SelectorPageTbl 处， 作用是让 es:edi 指向页表段的首地址， 即地址 PageTblBase 处，也是第一个页表的首地址。

在 bochs:10 处运行到 mov ecx, 1024\*1024 处，作用是让循环进行 1024 次，完成所有页表中 PTE

的初始化。

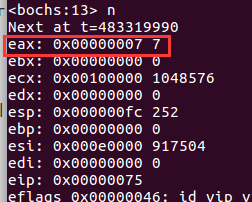


图 11: 页表初始化 3

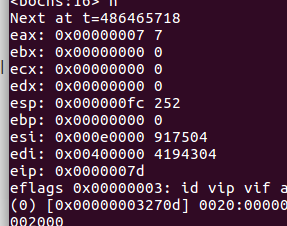


图 12: 页表初始化 4

在 bochs:13 处运行到 mov eax, PG\_P | PG\_USU | PG\_RWW 处， 作用是让第一个 PTE 赋值 为 PG\_P | PG\_USU | PG\_RWW，代表此 PTE 指示的页首地址为 0。

在 bochs:16 处运行到 loop .2 处，这里经过 1024\*1024 次循环，初始化所有 PTE，edi=1024\*1024\*4， 至此，页表的初始化完成。

6. 继续单步执行，启动分页机制:

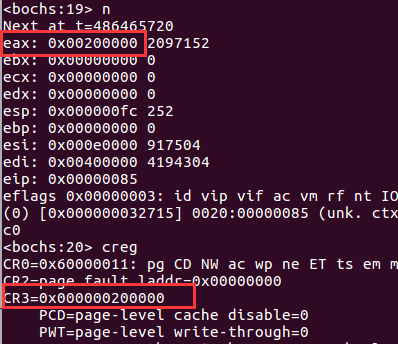


图 13: 分页 1

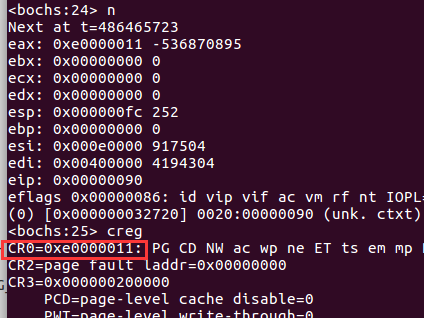


图 14: 分页 2

在 bochs:19 处运行到 mov eax, PageDirBase 处，作用是将 PageDirBase 的值赋给 eax 。 在 bochs:20 处运行到 mov cr3, eax 处，这里将 eax 的值赋给 cr3，使 cr3 指向页目标表。 在 bochs:25 处运行到 mov cr0, eax 处，设置 cr0 的 PG 位为 1 ，启动分页机制。

综上所述， 第一个循环 1024 次， 作用是完成页目录表中所有 PDE 的赋值， 即页目录表的初始 化， 这些 PDE 的属性相同， 都为指向可读可写的用户级别页表， 并且所有页表连续排列在以 PageTblBase 为首地址的 4MB 的空间中； 第二个循环 1024\*1024 次， 作用是完成页表中所有 PTE 的赋值， 即页表的初始化， 同样， 这些 PTE 的属性也相同， 都为可读可写的用户级别页， 循 环结束实现将 4GB 空间的线性地址映射到相同的物理地址的目的。

**2** **.** **2** **.** **3** 掌握 **PDE,PTE** 的计算方法

1. 动手画一画这个映射图 映射图如下图所示:

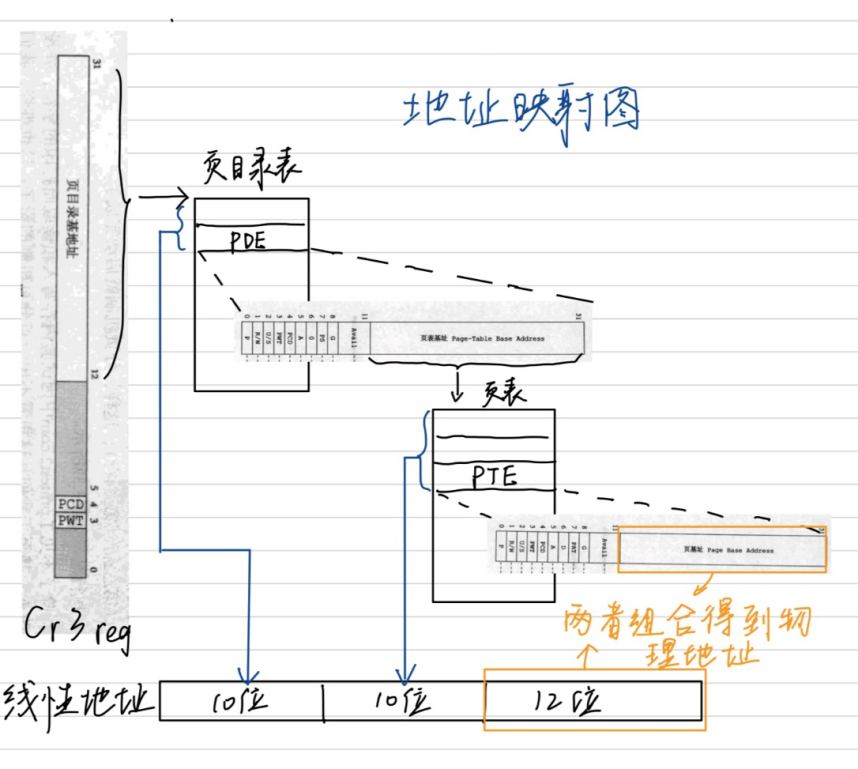


图 15: 映射图

2. 为什么代码 3.22 里面,PDE 初始化添加了一个 PageTblBase(Line 212), 而 PTE 初始化时候没有 类似的基地址呢(Line224)？

PageTblBase 的含义是页表基址， 需要给第一个页表初始化其首地址； 而 PTE 中， 第一个页表 的第一个 PTE 的页首地址直接指向 0 ，所以不用初始化类似的首地址。

**2** **.** **2** **.** **4** 熟悉如何获取当前系统内存布局的方法

利用中断 15h 知道机器的内存, 流程如下：

1. 在调用 15h 前, 先要做如下准备：

(a) 填充 eax=0E820h 。int 15h 的功能由 ax 决定, 如果想要得到内存信息, 需要给 ax 赋这样的

值。

(b) 填充 ebx=0 。第一次调用时 ebx 必须为零, 之后这个寄存器会放置后续值。

(c) 用 es:di 将其指向一个地址范围描述符结构 ARDS (Address Range Descriptor Structure) ,BIOS 会填充此结构。

(d) ecx ,es:di 所指向的地址范围描述符结构的大小, 以字节为单位。无论 es:di 所指向的结构如 何设置,BIOS 最多将会填充 ecx 个字节。不过, 通常情况下无论 ecx 为多大,BIOS 只填充 20 字节, 有些 BIOS 忽略 ecx 的值, 总是填充 20 字节。

(e) edx ,0534D4150h (SMAP ) BIOS 将会使用此标志, 对调用者将要请求的系统映像信息进行 校验, 这些信息会被 BIOS 放置到 es:di 所指向的结构中。

2. 中断调用后, 结果放在下面的寄存器中：

(a) CF ,CF=0 表示没有错误, 否则存在错误。

(b) edx ,0534D4150h (SMAP )。

(c) es:di , 返回的地址范围描述符结构指针, 和输入值相同。

(d) ecx ,BIOS 填充在地址范围描述符中的字节数量, 被 BIOS 所返回的最小值是 20 字节。

(e) ebx , 这里放置着为等到下一个地址描述符所需要的后续值, 这个值的实际形势依赖于具体的 BIOS 的实现, 调用者不必关心它的具体形式, 只需在下次迭代时将其原封不动地放置到 ebx

中, 就可以通过它获取下一个地址范围描述符。如果它的值为 0, 并且 CF 没有进位, 表示它 是最后一个地址范围描述符。

3. 地址范围描述符结构(Address Range Descriptor Structure)如下:

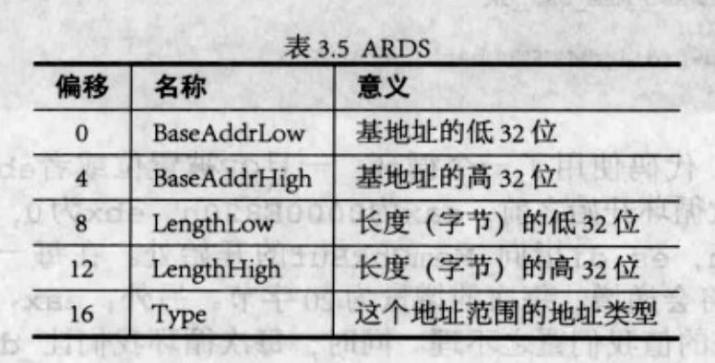


图 16: ARDS 结构

ARDS 中的 Type 取值如下:、

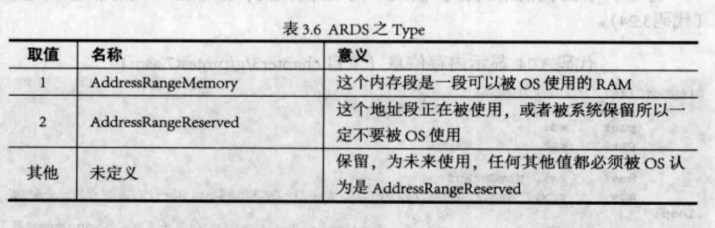


图 17: Type 取值

4.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

ax=0E820h 时调用 int 15h 得到的不仅仅是内存的大小, 还包括对不同内存段的一些描述。而且, 这些描述都被保存在一个缓冲区中。

在调用 int 15h 之前, 必须先有缓冲区。我们可以在每得到一次内存描述时都使用同一个缓冲区, 然后对缓冲区里的数据进行处理, 也可以将每次得到的数据放进不同的位置, 比如一块连续的内 存, 然后在想要处理它们时再读取。

后一种方式更方便一些, 最多可以存放 12 个 20 字节大小的结构体。

我们将把每次得到的内存信息连续写入这块缓冲区, 形成一个结构体数组。这个过程中,eax, ecx,

edx 的值都不会变,ebx 的值先不管,es:di 指向 \_MemeChkBuf 的开始处, 每次 di += 20 ,\_dwMCRNumber += 1 。最后当 CF 被置位或者 ebx 为零, 则结束循环。

课程代码中的一些体现:

~~ ~~

 DispMemSize : 

 push esi 

 push edi 

 push ecx 



 mov esi , MemChkBuf 

 mov ecx, [dwMCRNumber]*;for(int* *i=0;i<[MCRNumber];i++)//*每次得到一个*ARDS* 

 .loop : *;{* 

 mov edx , 5 *;* *for(int* *j=0;j<5;j++)* *//*每次得到一个*ARDS*中的成员 

 mov edi , ARDStruct *;* *{//*依次显示*BaseAddrLow,BaseAddrHigh,LengthLow,* 

 . 1 : *;* *LengthHigh,Type* 

 push dword [esi] *;* 

 call DispInt *;* *DispInt(MemChkBuf[j\*4]);* *//*显示一个成员 

 pop eax *;* 

 stosd *;* *ARDStruct[j\*4]* *=* *MemChkBuf[j\*4];* 

 add esi , 4 *;* 

 dec edx *;* 

 cmp edx , 0 *;* 

 jnz . 1 *;* *}* 

 call DispReturn *;* *printf("\n");* 

 cmp dword [dwType], 1*;* *if(Type* *==* *AddressRangeMemory)* 

 jne .2 *;* *{* 

 mov eax, [dwBaseAddrLow]*;* 

 add eax, [dwLengthLow]*;* 

 cmp eax, [dwMemSize] *;* *if(BaseAddrLow* *+* *LengthLow* *>* *MemSize)* 

 jb .2 *;* 

 mov [dwMemSize], eax *;* *MemSize* *=* *BaseAddrLow* *+* *LengthLow;* 

 .2 : *;* *}* 

 loop .loop *;}* 

 *;* 

 call DispReturn *;printf("\n");* 

32

33

34

35

36

37

38

39

40

41

42

43

|  |
| --- |
| push szRAMSize *;*  call DispStr *;printf("RAM* *size:");*  add esp , 4 *;*  *;*  push dword [dwMemSize] *;*  call DispInt *;DispInt(MemSize);*  add esp , 4 *;*  pop ecx  pop edi  pop esi  ret |

这段代码实际上是在访问 ARDS 数据结构的基础上获取物理内存布局信息。 ARDS 包含了系统内存中 每个物理内存段的描述, 包括其基地址、长度和类型等信息。通过遍历 ARDS 中的每个内存段, 并计算 其总长度, 代码可以确定系统的物理内存总大小。

**2** **.** **2** **.** **5** 掌握内存地址映射关系的切换

1. 画出流程图：

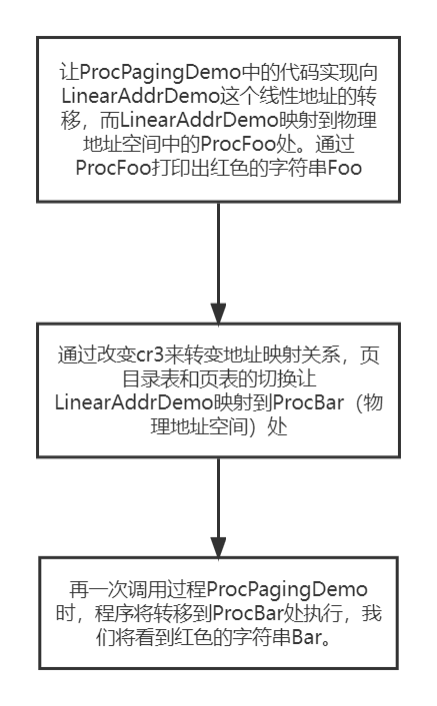


图 18: 内存地址映射关系

**2** **.** **2** **.** **6** 基础题

1. 依据实验的代码, 自定义一个函数, 给定一个虚拟地址, 能够返回该地址从虚拟地址到物理地址的 计算过程, 如果该地址不存在, 则返回一个错误提示。

(a) 解题方案与代码展示**:**

模拟 CPU 计算物理地址的过程, 逐步找到 PDE、PTE, 最终用 PTE 对应的物理页地址与页 内偏移合并即可。例：假设已知一个虚拟地址 0x01AF5518, 则转换的过程如下:

1

2

3

4

5

6

7

8

9

10

11

12





1

2

3

4





1

2

3

4

5

6

7

8

9

10

11

i. 把虚拟地址拆分成 3 个部分(低 12 位, 中 10 位, 高 10 位) , 拆分重新排列后： 页目录索引 00 000 00110, 页表项索引 10 1111 0101, 偏移 0101 0001 1000

二进制页目录索引 = 6, 页表项索引 = 0x2f5 , 偏移 = 0x518

相关代码如下:

|  |
| --- |
| mov  eax , VirtualAddr eax  DispInt  eax  ebx , eax  ecx , eax  edx , eax  ebx , 22  ecx , 10  ecx , 22  edx , 20  edx , 20  push  call  pop  mov  mov  mov  shr  shl  shr  shl  shr |

ebx、ecx、edx 分别对应页目录索引、页表项索引和偏移, 使用左右移运算得到需要的位。 使用 DispInt 函数打印。

ii. 根据当前的 CR3 寄存器中的物理地址定位页目录表基址

CR3 中存放的是物理地址, 这个物理地址指向进程的页目录表基址, 由此可以得到页目 录表基址 PDE = cr3 = 0xAA0E5000。

相关代码如下:

|  |
| --- |
| mov eax , PageDirBase1 *;*改为*cr3*结果一致  push eax  call DispInt  pop eax |

iii. 计算页表项的地址

页表地址存放在页目录表 PDE 中的第 6 个项目中, 也就是 [0xAA0E5000+4\*6]= [0xAA0E5018] = 0x3D955867, 其中 0x00000867 为该页表属性值, PTE = 0x3D955000。

相关代码如下:

|  |
| --- |
| shl ebx , 2  add eax , ebx  push eax  call DispInt  pop eax  mov eax, [ss:eax]  shr eax , 12  shl eax , 12  push eax  call DispInt  pop eax |

将 ebx 中的页目录索引左移两位即乘 4, 加上 eax 中存放的页目录表基址, 在目录表中对

应数据即页表项地址, 存在 ax 中, 再通过先右移再左移去除后 12 位的页表属性值, 得到 PTE 存在 eax 中, 并打印中间结果。

iv. 计算页面物理地址

要找的页面在这个页表中的第 0x2f5 项, 所以虚拟地址所在的页的物理地址为 [0x3D955000





1

2

3

4

5





1

2

3

4

1

2

3

4

5

6

7

8

9

10

11

+ 0x2f5 \* 4] = [0x3D955BD4] 假设 [0x3D955BD4] = 0x7095e847, 页面的物理地址 x0x7095e000, 0x00000847 表示的是页面属性。

相关代码如下:

|  |
| --- |
| ecx ,  shl add mov shr shl  2  ecx [ss:eax]  12  12  eax ,  eax ,  eax ,  eax , |

v. 计算最终的物理地址

由虚拟地址分离的偏移可以计算出最终的物理地址为： 0x7095E000 + 0x00000518 = 0x7095E518。用页面物理地址加上最开始分离出的偏移即可, 计算结束。

相关代码如下:

|  |
| --- |
| add eax , edx  push eax  call DispInt  pop eax |

vi. 错误提示

判断中间计算过程中的 PDE 和 PTE 计算值的合法性, 若不合法即跳转到.NO 开始执行 输出 NO 字符的代码。

相关代码如下:

|  |
| --- |
| mov eax, [ss:eax]  test eax , 0x00000001  jz .NO *;PDE*与*PTE*的合法性判别  .NO :  mov ah , 0Ch *;* *0000:* 黑底 *1100:* 红字  mov al , ’N’  mov [gs:((80 \* 20+ 0) \* 2)], ax *;* 屏幕第 *20*行*,* 第 *0*列。  al , ’O’  mov mov pop ret  [gs:((80 \* 20+ 1) \* 2)], ax *;* 屏幕第 *20*行*,* 第 *1*列。  ds |

(b) 结果展示: 在 pmtest8.asm 中直接添加该函数, 然后在 PSwitch 后 call 该函数即可。我们进

行了两次运行, 分别置虚拟地址 VirtualAddr 为 00401000h 和 99999999h, 输出不同的结果：

i. VirtualAddr 为 00401000h: 如下图, 输出了各种中间运算结果与最后的物理地址。

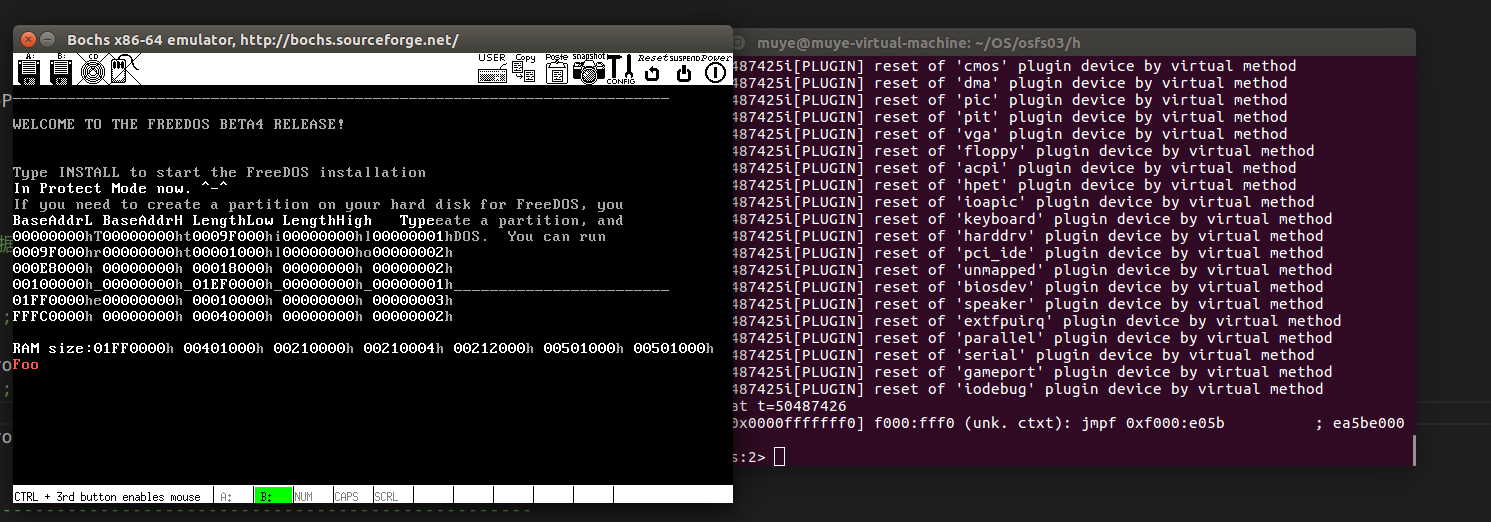


图 19: VirtualAddr 为 00401000h 的结果展示





1

2

3

4

5





ii. VirtualAddr 为 99999999h: 如下图, 由于该地址不存在, 运算过程中止, 并输出 NO。

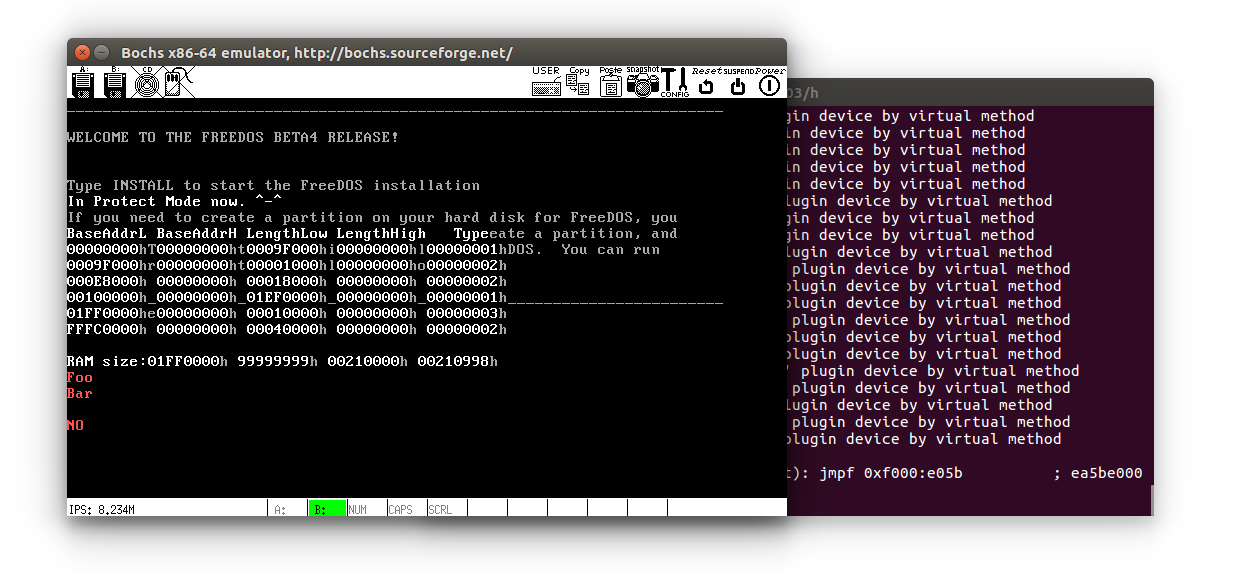


图 20: VirtualAddr 为 99999999h 的结果展示

2. 完善分页管理功能, 补充 alloc\_pages, free\_pages 两个函数功能, 试试你能一次分配的最大空间有 多大, 如果超出了有什么办法解决呢？

(a) 解题方案与代码展示**:**

i. 实现 alloc\_pages:

我们组实现 alloc\_pages 函数的目的重在建立线性地址和物理地址之间的映射， 并没过 分关注对于物理空间的管理，故只返回可用的物理地址。大致思路如下：

A. 寻找可用的物理页，计算并返回物理地址:

• 使用位图来查找可用的物理页:

|  |
| --- |
| \_BitMap : times 32db 0xff *;* *low* *1MB* *is* *occupied(0* *~* *0xfffff)*  times 32db 0x0 *;* *1MB* *is* *available(0x100000* *~* *0x1fffff)*  BitMap equ \_BitMap - $$  BitMapLenequ $ - \_BitMap |

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

这里在数据段定义了位图， 假设 0-1MB 物理内存被占用， 1MB-2MB 处的物理内存未被占用。

• 编写 alloc\_a\_4k\_page 函数计算并返回物理地址:

~~ ~~

 alloc\_a\_4k\_page : *;* *arg* *none* 

 *;* *return* *eax:* *physical* *address* 

 *;* *physical* *address* *begin* *at* *0x00000000* 

 *;* *save* *reg* 

 push ds 

 push es 

 *;* *fuction* *implement* 

 xor eax , eax 

 mov ax , SelectorFlatRW 

 mov es , ax 

 mov ax , SelectorData 

 mov ds , ax 



 .search : 

 bts [BitMap], eax 

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

|  |
| --- |
| jnc .find  inc eax  cmp eax ,BitMapLen\*8  jl .search  *;* *no* *available* *physical* *space*  *;* *we* *should* *move* *one* *page* *back* *to* *disk*  *;* *but* *for* *simplicity,* *we* *hlt*  hlt  .find :  shl eax , 12 *;ecx* *->* *avaiable* *physical* *addr*  pop es  pop ds  ret |

函数首先保存寄存器的值， 将选择子加载到段寄存器中， 然后使用位图来查找可用的物理页， 具体而言， 使用 bts 指令在位图 BitMap 中设置位， 表示对应的物理页已被分配。然后， 使用 jnc 指令判断设置位的结果， 如果未设置成功， 则跳转到标签.ﬁnd。如果设置成功， 则将 eax 左移 12 位，将其作为可用的物理地址， 并通过 pop 指令恢复寄存器的值， 最后使用 ret 指令返回物理地址。在标签.ﬁnd 处， 将 eax 左移 12 位，将其作为可用的物理地址，并通过 pop 指令恢复寄存器的值，最后使用 ret 指令返回物理地址。如果没有可用的物理空间，则会执行 hlt 指令暂停程序的执行。

值得一提的是， 如果没有可用的物理空间， 可以采用虚拟内存的方式， 将部分数据存在磁盘上，但这里为了简化代码，使用 hlt 指令，直接暂停程序执行。

B. 完成虚拟地址对物理地址的映射关系:

• 基于当前 PDE 和 PTE 找到当前页表。

• 基于页表项的 p 标志判断页表是否存在，不存在则创建新页表，并与页目录项进行关联。

具体代码如下:

~~ ~~

 alloc\_pages : *;* *arg* *:* *eax* *:* *page* *number* 

1

2

3

4

5

6

7

8

9

10

 *;* *return* *ebx* *:* *linear* *address* 

 *;* *save* *registers* 

 push ds 

 push es 

 *;* *function* *implement* 

 mov bx , SelectorFlatRW 

 mov ds , bx 

 mov bx , SelectorData 

 mov es , bx 

11

 mov ecx , eax *;* *ecx* *means* *the* *number* *of* *page* 

12

13

14

 mov ebx , 4096 

 mul ebx *;* *ebx* *means* *the* *size* *of* *pages* 

15

 mov ebx, [es:AvaLinearAddress] *;* *ebx* *means* *the* *return* *value* 

16

17

18

19

20

21

 add [es:AvaLinearAddress],eax *;* *update* *the* *addresss* *of* *free* *linear* *address* 

 push ebx *;* *save* *the* *return* *value* 

 mov eax , ebx 

 mov ebx , cr3 *;* *assume* *linear2Physical(cr3)* *=* *cr3* 

 and ebx , 0xfffff000 

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

|  |
| --- |
| and eax , 0xffc00000  shr eax , 20  add ebx , eax *;* *ebx* *means* *the* *pde* *item*  mov edx , ebx *;* *edx* *->* *missing* *pde* *address*  mov ebx, [ebx] *;* *ebx* *means* *the* *corresponding* *page* *table* *item*  *;* *test* *p* *flag*  test ebx , 0x0000\_0001  jnz .pde\_exist  *;* *page* *table* *is* *not* *exist*  mov ebx , cr3  mov ebx, [ebx] *;* *first* *page* *table* *address*  and ebx , 0xfffff000  shl eax , 10*;* *eax* *means* *the* *size* *of* *used* *pages*  add ebx , eax  or ebx , 0x0000\_0007  mov [edx], ebx |

使用 alloc\_a\_4k\_page 函数分配单个 4KB 的页，并设置对应页表项。

•

循环执行前面步骤，完成所有页的分配。

•

具体代码如下, 标签.pde\_exist 表示页表项存在的情况， 标签.change\_pte 表示循 环修改页表项的过程。

|  |
| --- |
| .pde\_exist :  mov eax,[esp]  and ebx , 0xfffff000  and eax , 0x003ff000  shr eax , 10  add ebx , eax  .change\_pte :  call alloc\_a\_4k\_page  or eax , 0x00000007  mov [ebx] , eax  add ebx , 4  loop .change\_pte  pop ebx *;set* *return* *value*  pop es  pop ds  ret |

实现 free\_pages:

ii.

要完成释放已分配的页， 只需要修改对应页目录项和页表项， 取消虚拟地址到物理地址 的映射关系即可，具体实现代码如下：

~~ ~~

 free\_pages : *;* *arg* *eax,* *linear* *address* *,* *ebx* *page* *number* 

1

2

3

4

5

 push ds 

 push es 

 push ebx *;* *save* *eax* *and* *ebx* 

 push eax 

6

 mov bx , SelectorFlatRW 

7

8

9

10

 mov ds , bx 

 mov bx , SelectorData 

 mov es , bx *;* *normal* *init* 

11

 *;* *find* *the* *pde* *and* *pte* 

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44





1

2

3

4

5

6

7

8

9

10





|  |
| --- |
| mov ebx , cr3  and ebx , 0xfffff000  and eax , 0xffc00000  shr eax , 20*;* *20=* *22* *-2*  add ebx , eax *;* *ebx* *now* *means* *the* *pde* *item*  mov edx, [ebx]  and edx , 0xfffffff8  mov [ebx], edx *;* *set* *the* *final* *3* *-bit* *zero* *and* *store* *back*  mov ebx, [ebx] *;* *now* *ebx* *means* *the* *first* *pte* *item*  mov eax, [esp] *;* *now* *eax* *is* *the* *liner* *address*  add esp , 4  and ebx ,0xfffff000  and eax ,0x003ff000  shr eax , 10  add ebx , eax *;* *now* *ebx* *means* *the* *right* *pte* *item*  mov ecx, [esp] *;* *here* *ecx* *means* *page* *number*  add esp ,4  .change\_pte : *;* *set* *every* *item’s* *last* *3* *-bit* *zero*  mov eax, [ebx]  and eax , 0xfffffff8  mov edx , eax *;* *now* *eax* *is* *the* *physical* *address*  shr edx , 12  btr [BitMap], edx  mov [ebx], eax  add ebx ,32  loop .change\_pte  pop es  pop ds  ret |

具体来说， 函数先获取 cr3 寄存器的值， 并将其与位掩码进行与运算， 得到页目录项 (PDE) 的地址后修改 PDE 项的值， 将最后三位设置为零， 并将修改后的值存回内存。接着， 将传入的物理地址进行位运算， 计算出对应的页表项(PTE) 的地址后使用循环遍 历每个 PTE 项， 将每个项的最后三位设置为零， 并将修改后的值存回内存。完成释放已 分配的页的作用。

iii. 编写测试函数查看结果:

|  |
| --- |
| TestAllocAndFree :  xchg bx ,bx  mov eax ,4  call alloc\_pages  xchg bx ,bx  mov eax ,ebx  mov ebx ,4  call free\_pages  xchg bx ,bx  ret |

(b) 结果展示**:**

i. cr3=0x000000200000 情况下：

A. 查看 alloc\_pages 之前地址的映射关系:

E:\Desktop\OS实验\os3\实验截图\实验内容\结果1.png结果1

图 21: 结果 1

B. 查看 alloc\_pages 之后地址的映射关系:

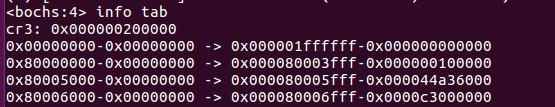


图 22: 结果 2

C. 查看 alloc\_pages 之后地址的映射关系:

E:\Desktop\OS实验\os3\实验截图\实验内容\结果3.png结果3

图 23: 结果 3

ii. cr3=0x000000210000 情况下：

A. 查看 alloc\_pages 之前地址的映射关系:

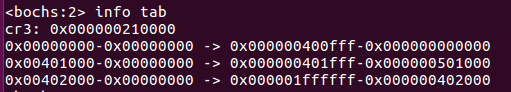


图 24: 结果 4

B. 查看 alloc\_pages 之后地址的映射关系:

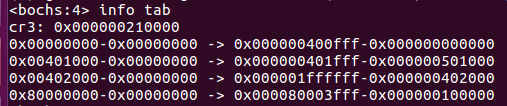


图 25: 结果 5

C. 查看 alloc\_pages 之后地址的映射关系:

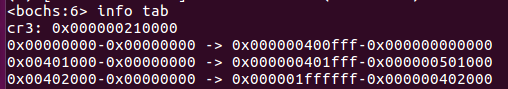


图 26: 结果 6

可以看到在 cr3=0x000000200000 情况下， alloc\_pages 前只有 1 个映射关系,在调用 alloc\_pages 后有了4 个映射关系, 在 free\_page 后, 又变成了1 个对应关系。在 cr3=0x000000210000 情况下， alloc\_pages 前只有 3 个映射关系, 在调用alloc\_pages 后有了 4 个映射关系, 在 free\_pages 后, 又变成了 3 个对应关系。alloc\_pages 函数和 free\_pages 函数编写成功。

如果分配的空间超出了系统的物理内存大小， 可以采用虚拟内存的方式， 将部分数据存储在磁盘上， 进而实现扩展可用内存的大小， 当需要这些数据时， 将数据从磁盘加载到物理内存中。

三、 实验过程分析

（实验分工， 详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格等）

1. 在设置断点后程序仍然一次性执行完成:

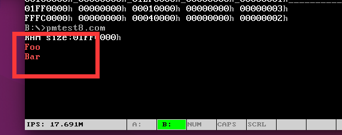


图 27: 错误 1

解决： 在翻阅资料之后发现首先需要对 bochsrc 配置文件进行修改。而不能只在程序中相关位置 添加 xchg bx,bx

2. 基础题 1 的理解不明确:

(a) 对地址不存在的条件判别不到位:

解决: 在编写.NO 的代码时, 一开始只考虑到对物理地址的检查， 单纯同 ProcBar 的值相比 较， 实际上逻辑上发生了错误， 理应是检查页目录和页表编号的合法性。原先的理解导致程 序始终无法跳转到.NO 并输出 NO 字符。

(b) 忽视了页目录表属性和页面属性位:

解决: 在计算物理地址过程中， 并没有考虑到虚拟地址中的页目录表属性和页面属性位， 造 成物理地址的计算错误。下图是错误展示， 代码修改后使用重复的左移和右移以获取准确的 页目录和页面信息。

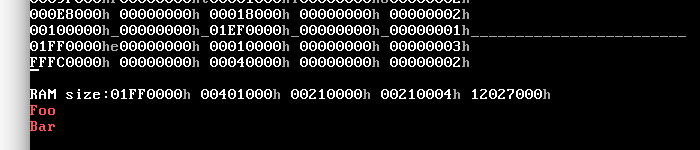


图 28: 错误 2

四、 实验结果总结

对实验结果进行分析*,* 完成思考题目*,* 并提出实验的改进意见

**4** **.** **1** 思考题

**4** **.** **1** **.** **1** 分页和分段有何区别？在本次实验中**,** 段页机制是怎么搭配工作的？

1. 页是信息的物理单位， 分页是为实现离散分配方式， 以消减内存的外零头， 提高内存的利用率； 段 是信息的逻辑单位， 含有一组意义相对完整的信息。分段机制是将逻辑地址转换成线性地址， 分 页机制则是将线性地址转换成物理地址。

2. 在本次实验中 GDT 定义了两个段 descriptor 分别用来存放目录表和页表。在 pmtest6.asm 中， 页目录表为 4KB，页表为 4MB 。PageDirBase 和 PageTblBase 是两个宏， 指定了页目录表和页 表在内存中的位置。在段中先对 PDE 、PTE 、cr0 和 cr3 进行初始化，从而实现了分页机制。

**4** **.** **1** **.** **2** **PDE** 、**PTE,** 是什么？例程中如何进行初始化？ **CPU** 是怎样访问到 **PDE** 、**PTE,** 从而计

算出物理地址的？

1. PDE 、PTE 是什么？

PDE (Page Directory Entry) 和 PTE (Page Table Entry) 是 x86 架构的操作系统中重要的数 据结构。 PDE 是页目录项， 每个 PDE 包含一个指向页表的物理地址以及一些控制位， 如权限位、 存在位等。一个页目录可以映射 4MB 的虚拟地址， 即包含 1024 个 PDE 。PTE 是页表项， 每个 PTE 包含一个指向实际物理页的物理地址， 以及一些控制位， 如权限位、存在位等。同样， 一个页表可以映射 4KB 的虚拟地址空间，即包含 1024 个 PTE。

2. 例程中如何进行初始化？

(a) 创建一个页目录(Page Directory)和多个页表(Page Table)。

(b) 将页目录和页表的物理地址放入 CPU 的控制寄存——CR3 寄存器。

(c) 对每个需要映射的虚拟地址区域，设置相应的 PDE 和 PTE ，包括权限位、存在位等。

3. CPU 如何访问 PDE 和 PTE 以计算物理地址？

(a) 当一个程序访问虚拟地址时， CPU 首先将该虚拟地址拆分为三个部分： 页目录索引、页表索

引和偏移。

(b) 使用页目录索引， CPU 访问当前进程的页目录，找到对应的 PDE。

(c) 如果 PDE 中的存在位有效(表示此 PDE 映射了一个页表)，则使用页表索引， 访问相应的 页表。

(d) 在页表中， CPU 找到相应的 PTE ，检查存在位是否有效。

(e) 如果 PTE 的存在位有效， CPU 使用 PTE 中的物理地址和偏移来计算最终的物理地址。

**4** **.** **1** **.** **3** 开启分页机制之后**,** 在 **GDT** 表中、在 **PDE** 、**PTE** 中存的地址是物理地址、线性地址**,** 还

是逻辑地址**,** 为什么？

GDT (Global Descriptor Table)、PDE (Page Directory Entry)、PTE (Page Table Entry) 中存 储的地址是物理地址。存储物理地址才能正确定位到段、页的位置， 线性地址和逻辑地址是用于实现虚 拟内存而使用的，但 GDT 、PDE 、PTE 本身存储的仍然是物理地址。

**4** **.** **1** **.** **4** 为什么 **PageTblBase** 初始值为 **2M+4K** ？能不能比这个值小？

因为设置页目录表(Page Directory Table) 的起始地址为 2M, 而页目录表大小为 4K, 那么 PageT- blBase 的初始值必须确保不会与页目录表的位置发生交叉。如果 PageTblBase 的初始值小于 2M+4K, 会导致页表项(Page Table Entry,PTE) 与页目录表项(Page Directory Entry,PDE) 存放位置交叉, 从 而导致内存管理错误。

**4** **.** **1** **.** **5** 怎么读取本机的实际物理内存信息？

利用中断 15h。先填充如下寄存器：

• CF: CF=0 表示没有错误，否则存在错误。

• eax: 0534D4150h (’SMAP’)。

• es: di : 返回的地址范围描述符结构指针，和输入值相同。

• ecx: BIOS 填充在地址范围描述符中的字节数量，被 BIOS 所返回的最小值是 20 字节。

• ebx: 这里放置着为等到下一个地址描述符所需要的后续值， 这个值的实际形势依赖于具体的 BIOS 的实现， 调用者不必关心它的具体形式， 只需在下次迭代时将其原封不动地放置到 ebx 中， 就可以通过它获取下一个地址范围描述符。如果它的值为 0，并且 CF 没有进位， 表示它是最后

一个地址范围描述符。



图 29: 地址范围描述符结构

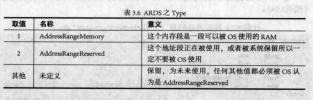


图 30: Type 取值

由上面的说明， ax=0E820h 时调用 int 15h 得到的不仅仅是内存的大小， 还包括对不同内存段的一些描述。而且， 这些描述都被保存在一个缓冲区中。所以， 在我们调用 int 15h 之前， 必须先有缓冲区。我们可以在每得到一次内存描述时都使用同一个缓冲区， 然后对缓冲区里的数据进行处理， 也可以将每次得到的数据放进不同的位置， 比如一块连续的内存， 然后在想要处理它们时再读取。

**4** **.** **1** **.** **6** 如何进行地址映射与切换？

进程的虚拟地址需要首先通过其局部段描述符变换为 CPU 整个线性地址空间中的地址， 然后再使 用页目录表和页表映射到实际物理地址页上。具体方法为由寄存器 CR3 指定的页目录中根据线性地址 的高 10 位得到页表地址， 然后在页表中根据线性地址的第 12 到 21 位得到物理页首地址， 将这个首地 址加上线性地址低 12 位便得到了物理地址。

在 pmtest8.asm 中，一开始， 调用 ProcPagingDemo ，向 LinearAddrDemo 这个线性地址进行跳转， 而 LinearAddrDemo 映射到物理地址空间的 ProcFoo 处， 我们让 ProcFoo 打印出红色的字符串 Foo ， 所以执行时我们应该可以看到红色的 Foo 。随后我们改变映射关系， 实际上就是改变页目录表的指向， 页目录表和页表的切换将 LinearAddrDemo 映射到 ProcBar 处， 这时我们调用 ProgPagingDemo 时，程序转移到 ProcBar 处执行，从而打印出红色字符串 Liu。

**4** **.** **1** **.** **7** 如何实现 **alloc\_pages** **,** **free\_pages**

详见基础题 2

**4** **.** **2** 对实验的改进意见

**4** **.** **2** **.** **1** 程子洋

1. 希望基础题可以添加具体的指示, 题目要求可以更加清晰。

2. 提供学生在遇到常见问题时进行自我排除的指南。这可以包括常见错误消息的解释以及如何解决 这些问题的步骤。

**4** **.** **2** **.** **2** 聂森

1. 可以先让同学阅读教材资料后再介绍实验内容和实验相关的知识， 可以帮助同学更快更好的了解 实验的目标和实验的原理及任务。

2. 给出具体的实验指导获给出更加丰富的实验参考资料， 可以帮助同学更快上手实验并在有问题时 找到解决方式。

**4** **.** **2** **.** **3** 王卓

1. 提供更详细的实验指导： 在每个实验步骤中， 提供更详细的指导和说明， 比如预期结果等， 这样 能帮助我们更好地完成实验。

2. 介绍实验目的和背景： 实验开始之前提供实验的目的和背景， 解释一下为什么需要进行该实验以 及其与操作系统的关系，这能帮助我们更好理解实验的意义和重要性。

**4** **.** **2** **.** **4** 刘虓

希望能过教学示例，学生自己操作时可以更快上手。

五、 各人实验贡献与体会

每个人分别填写自己在本次实验中的分工*,* 并总结实验的心得体会。

**5** **.** **1** 程子洋

**5** **.** **1** **.** **1** 分工

在本次实验中， 本人完成大部分实验内容， 负责实验内容第 5 题、基础题第 1 题、思考题第 4 题 实验报告的撰写与实验报告的最终整合。

**5** **.** **1** **.** **2** 心得体会

这个实验涉及到复杂的内存管理和汇编编程概念， 需要仔细阅读相关文档和章节资料， 同时也需 要通过实际操作来加深理解。实验的思考题部分可以帮助我更深入地思考和理解内存管理机制， 能够 通过这个实验学到很多有关操作系统内存管理的知识。在实验中， 需要理解分页和分段的方法如何 结合使用，以便在保护模式下进行内存管理。

由于我对汇编不是特别熟练、对本章知识不是完全理解等等原因， 编写和调试还是花了很长时间， 中间出现了各种奇怪的错误， 不过最终还是得到了正确的结果， 感觉自己收获颇丰， 极大地加深了我对 分页机制和汇编编写的理解。

**5** **.** **2** 聂森

**5** **.** **2** **.** **1** 分工

在本次实验中， 本人完成大部分实验内容， 负责实验内容的第 3 题、思考题的第 2 题、第 3 题的 实验报告撰写。

**5** **.** **2** **.** **2** 心得体会

在本次实验中， 我对上学期学习的虚拟内存技术有了更加深刻的认识。原本认为很难的虚拟内存 实现， 在仔细阅读教材和调试之后发现并没有那么困难， 并在实验的过程中增强了理论实现的能力。对 于 PDE 和 PTE 等相关知识，在实验中也逐步清晰和掌握，通过对实验代码的调试和学习，也使我掌 握了内存映射切换的方法。实验过程中有些许磕磕绊绊， 由于汇编知识和 OS 知识的遗忘， 实验也不得

不放缓速度以先复习相关内容再开展实验， 但也正是这些过程才让我对知识掌握的更加扎实。希望自 己能在实践中学习更多!

**5** **.** **3** 王卓

**5** **.** **3** **.** **1** 分工

在本次实验中， 本人完成大部分实验内容， 负责实验内容的第 1 题、第 2 题、基础题第 2 题、思 考题的第 1 题的实验报告撰写。

**5** **.** **3** **.** **2** 心得体会

本次实验起初有点困难, 因为虽然在上学期学习过段页式, 但是对线性地址转换成物理地址等等操 作还是在很多细节上出现了问题。比如在代码编写中， 对地址的转换不熟悉， 忘记了需要右移 22 位后 再左移 4 位， 导致 debug 时间很长， 所以需要在实验前进一步熟悉理论。但更多的是， 在实验过程中

的不断试错， 也让我逐步清晰和掌握了对于 PDE 和 PTE、物理内存信息的读取以及内存的映射切换 等相关知识。希望在后续实验中能进一步巩固汇编语言的知识，同时加深对操作系统的理解！

**5** **.** **4** 刘虓

**5** **.** **4** **.** **1** 分工

在本次实验中， 本人完成大部分实验内容， 负责实验内容的第 5 题、思考题的第 5 题、第 6 题的 实验报告撰写。

**5** **.** **4** **.** **2** 心得体会

通过本次实验，我对 80x86 的内存管理机制有了更深刻的理解， 对于地址映射和跳转也更加熟悉。 同时， 与小组成员的分工合作也让我意识到团队的重要性， 多人的智慧与努力在攻克问题时的效率远 不是单干能比的。总而言之， 本次实验让我不仅对分页机制都有了实践意义上的认知， 更对汇编编程也 有了更深的理解，我从中获益匪浅。

六、 教师评语

|  |  |  |
| --- | --- | --- |
| 教师评语 | | |
|  | | |
| 姓名 | 学号 | 分数 |
| 程子洋 | 2021301051114 |  |
| 聂森 | 2021302191536 |  |
| 王卓 | 2021302191791 |  |
| 刘虓 | 2021302121234 |  |
| 教师签名：  年 月 日 | | |