|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 | | 2021.10.8 |
| 实验名称 | 段页式实验 | | 实验周次 | | 第三周 |
| 姓名 | 学号 | | 专业 | | 班级 |
| 鄢锦琪 | 2019302180149 | | 信息安全 | | 19级大类五班 |
|  |  | |  | |  |
|  |  | |  | |  |
|  |  | |  | |  |
| 1. 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） | | | | | |
| 实验目的: 掌握分页机制的相关内容  掌握的知识: 掌握内存分页机制  实验内容: 对应参考阅读章节：第三章3.3节  参考代码: 基本分页机制/f/pmtest6.asm  读取内存，并合理填充/g/pmtest7.asm  体会分页， /h/pmtest8.asm | | | | | |
| 1. 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） | | | | | |
| 实验环境:Ubuntu14.04 , virtual box , bochs2.7.0  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 – 动手画一画这个映射图 – 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：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。  – 完善分页管理功能，补充alloc\_pages, free\_pages两个函数功能  思考的问题: 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. 为什么PageTblBase初始值为2M+4K？能不能比这个值小？  4. 怎么读取本机的实际物理内存信息？  5. 如何进行地址映射与切换？  6. 如何实现alloc\_pages,free\_pages | | | | | |
| 1. 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） | | | | | |
| 1.什么是分页机制 ? 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   * 分页机制在段机制之后进行的，它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, 将线性地址通过某种变换映射到物理地址中. * (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。段则是信息的逻辑单位，它含有一组其意义相对完整的信息。分段的目的是为了能更好地满足用户的需要。 * (2) 页的大小固定且由系统决定；而段的长度却不固定，决定于用户所编写的程序。 * (3) 分页的地址空间是一维的，程序员只需利用一个记忆符，即可表示一个地址；而分段的作业地址空间是二维的，程序员在标识一个地址时，既需给出段名，又需给出段内地址。 * 段页机制在此处是, 通过将代码分为不同段, 可以给不同段的代码赋予不同的权限和属性, 比如在此处就定义了数据段,代码段, 非一致代码段和一致代码段等等不同权限和属性的段, 其中包含数据或者是代码。而分页则是用于为了更好的利用存储空间，提高内存的利用率。在本次实验中， 就通过段机制区分不同功能的代码和数据，在GDT中定义了两个段，分别存放页目录和页表。 然后利用分页机制分配内存   2.代码3.22中的212--237行的几个循环做了什么 ? 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   * 循环1的作用是初始化页目录, 首先设循环次数为1024次, 给eax赋初值后进入循环,通过stosd指令将eax 存储到es:edi指向的内存单元中(同时edi根据DF寄存器进行变化), 因为一个页的大小是4k , 所以地址相对要增加4k , 将eax增加4096 , 也就得到了下一个表的地址, 也就是页目录表中的下一项, 然后通过循环的方式得到了一个页目录表, 打断点查看内存, 打断点处如图所示 * 因为将eax存在了es:edi位置, 所以查看相应内存,首先查看在进入循环前时es:edi指向的位置, 显示是0x00200000,此时的eax是0x00201007, 使其完成循环后查看内存.正如上面所说 * 循环2的作用是初始化所有页表, 与循环一同理 ,在循环1结束后的es:edi是0x00201000, 使其循环结束后,查看内存如下 * 因为此处的地址映射是f(x) = x, 所以第一项对应0x00000007(最后三位是全1),后面每项也符合要求.符合上面所说 * 页目录的表项简称PDE(Page Directory Entry) * 页表的表项简称PTE (Page Table Entry) * 在代码3.22中, 首先定义页目录和页表的开始地址, 并且开辟两个新的段, 定义相应的描述符,选择子。此处pmtest6.asm中，采用的是将线性地址映射到真实的物理地址中，并且是连续排布， 所以初始化就是直接计算出相应的线性地址，然后通过循环的方式，连续存入表项。 * CPU先是从寄存器cr3指定的页目录中根据现行地址的高10位得到页表地址，然后再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。   3.PDE，PTE的计算方法   * cr3 中读取到页目录表的起始地址, 然后结合线性地址的前10位获得页表的起 * 始地址, 结合中间10位获得物理页的起始地址, 最后12位作为偏移, 直接在物理页上偏移, 最终即是物理地址 * 未命名绘图 * 因为在代码3.22中的地址的映射是f(x) = x, 线性地址和物理地址相同 , 所以物理地址从0开始, 基地址就是0 , 所以没有看起来像初始化pde时的PageTblBase   4.为什么PageTblBase初始值为2M+4K？能不能比这个值小？   * 因为设置页目录表起始位置为2M，然后页目录表占4K，然后页目录表与页表在内存中相邻，所以是页表的初始值是2M+4K。 * 能不能比2M+4K小取决于页目录表的起始地址，页目录表的大小和页的大小，只要减少页目录项,或减小页大小，完全可以小于2M+4k。比如在后续的代码中，因为提前得知了内存的大小， 也就不需要那么多页目录和页表，实际的页表基地址就小于2M+4K   5.怎么读取本机的实际物理内存信息？   * 采用BIOS中断中的：int15h。 * 在调用中断15h之前，需要填充如下寄存器：   + eax int 15h可完成许多工作，主要由ax的值决定，我们想要获取内存信息，需要将ax赋值为0E820h。 ebx 放置着“后续值（continuation value）”，第一次调用时ebx必须为0。   + es:di 指向一个地址范围描述符结构ARDS（Address Range Descriptor Structure），BIOS将会填充此结构。 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。   + 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。 中断调用之后，结果存放于下列寄存器之中。   + 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。 * 循环读取结构体ARDS，就能计算出本机中可使用的物理内存， 并且根据该内存可以计算出所需要的页数和页表数 * **此处的中断是在实模式下调用的 ，因为保护模式下的中断机制需要额外配置。并且在实模式下访问内存可以直接用标号， 但是在保护模式下应该使用偏移**   6.如何进行地址映射与切换（画出流程图）？   * 地址映射，CPU先是从寄存器cr3指定的页目录中根据现行地址的高10位得到页表地址，然后再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。 * 地址切换，重新初始化一个新的页目录表和页表，然后将这个页目录表的首地址经过处理放进cr3寄存器中，然后按照上面地址映射的同样方法进行映射，就得到了新的物理地址   7.自定义一个函数，给定一个虚拟地址能够返回物理地址，若不存在，则返回一个错误提示  Linear2Physical:  push ds  mov bx, SelectorFlatRW  mov ds,bx  mov ebx,cr3  and ebx,0xfffff000   ;push eax  ;call DispInt  ;add esp,4   mov ecx, eax  shr eax,22  and eax,0x3ff  shl eax,2  mov ebx,[eax+ebx]    test ebx,0x00000001  jz .not\_exist    and ebx, 0xfffff000    mov eax, ecx  shr eax,12  and eax,0x3ff  shl eax,2  mov ebx,[eax+ebx]    test ebx,0x00000001  jz .not\_exist    and ebx,0xfffff000   and ecx,0xfff  add ebx,ecx  mov eax,ebx  jmp .exit .not\_exist:  mov ebx,0xffffffff .exit:  pop ds  ret  eax作为线性地址的输入 , ebx作为物理地址的输出,若地址不正确则返回全1表示出现异常。  试图在该函数中直接使用lib.inc中实现的Dispint和DispStr打印中间过程，不过在实操后发现在调用Dispint等时， ds会发生变化，所以并不方便，于是选取和助教相同的直接打印开始和结果  TestL2P:  push ds  push edx  mov dx, SelectorData  mov ds,dx   push eax  push Linear  call DispStr  add esp,4  pop eax   push eax  call DispInt  pop eax  call Linear2Physical   push eax  push Physical  call DispStr  add esp,4  pop eax   push ebx  call DispInt  pop ebx   pop edx  pop ds  ret  然后在进行页转换前后进行测试    8.完善分页管理功能,补充alloc\_pages 和 free\_pages 两个函数功能.  首先写一个分配4k的页的函数,没有输入, 输出是eax,存储着分配后可用的物理地址。  alloc\_a\_4k\_page: ; arg none  ; return eax: physical address  ; physical address begin at 0x00000000  push ds  push es    xor eax, eax  mov ax, SelectorFlatRW  mov es, ax  mov ax, SelectorData  mov ds, ax   .search:  bts [BitMap], eax  jnc .find  inc eax   cmp eax,BitMapLen\*8  jl .search  hlt  .find:   shl eax,12  pop es  pop ds  ret  然后写一个分页的函数, 输入值 是eax ,存储着页的数量, ebx存储着线性地址  alloc\_pages: ; arg: eax : page number  ; return ebx : linear address   ; save registers   push ds  push es    mov bx, SelectorFlatRW  mov ds, bx  mov bx, SelectorData  mov es, bx    mov ecx, eax ; ecx means the number of page   mov ebx, 4096  mul ebx ; ebx means the size of pages    mov ebx, [es:AvaLinearAddress] ; ebx means the return value  add [es:AvaLinearAddress],eax ; update the addresss of free linear address  push ebx ; save the return value  mov eax, ebx  mov ebx, cr3  and ebx, 0xfffff000  and eax, 0xffc00000  shr eax, 20  add ebx, eax ; ebx means the pde item  mov edx, ebx  mov ebx, [ebx] ; ebx means the corresponding page table item    test ebx, 0x0000\_0001  jnz .pde\_exist    mov ebx, cr3   mov ebx, [ebx]   and ebx, 0xfffff000   shl eax, 10 ; eax means the size of used pages  add ebx, eax  or ebx, 0x0000\_0007  mov [edx], ebx  .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  pop es  pop ds  ret  该函数就是按照线性地址映射到物理地址的方式,先找到PDE，判断PDE是否合法，若不合法则结束，若合法则继续找到PTE， 然后根据页数初始化对应个数的PTE。  free\_pages: ; arg eax,linear address , ebx page number  push ds  push es  push ebx ; save eax and ebx  push eax   mov bx, SelectorFlatRW   mov ds, bx  mov bx, SelectorData  mov es, bx ; normal init    ; find the pde and pte   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  此处的free\_page类似于alloc\_page, 只是这里不需要判断是否PDE和PTE是否合法, 因为,我们就是要将其变成不可用, 若本就不可用我们再将其变成不可以也没有影响。根据线性地址和CR3找到对应的PDE ， 将其置为不可用，然后继续根据线性地址的中间10位找到PTE， 将规定页数的PTE也置为不可用。然后在找到了相应页的物理地址后,(将地址除以4096) 还需要将Bitmap中的相应位置为0, 此处使用btr指令,将其置0  测试函数如下  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  测试结果如下    可以看到在alloc\_page前只有三个映射关系,在调用alloc\_page后有了4个映射关系,在free\_page后,又变成了3个对应关系。 | | | | | |
| 1. 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） | | | | | |
| 实验结果:  实验结果如上节所示, 完成从线性地址到物理地址的函数, 完善了分页机制,完成了alloc\_page和free\_page两个函数  运行结果如下图所示      本次实验的不足:  1、更熟悉理论再编码。本次实验中起初有点困难, 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。比如在编写Linear2Physical时，对地址转换不熟，忘记了需要在右移22位后再将其左移四位，于是出现很多bug，最后debug花费了很长时间，最后参考了助教的代码才知道问题所在， 所以下次实验时要将理论先熟悉最后再开始敲代码。  2、要有大局观，在调用dispInt和dispStr的时候。有时候会遇到直接程序崩溃。后来在debug时发现，是因为在dispInt的时候要用到数据段的一个变量dWDispPos，但是这个函数默认ds是数据段，而在Linear2Physical中，因为将ds置为SelectorFlatRW了，是没法直接访问dWDispPos位置的，所以就会出现问题，于是没有在Linear2Physical函数中直接调用DispInt和DispStr函数。同样因为汇编语言中的通用寄存器一般就使用eax, ebx, ecx, edx但是在函数中往往四个变量并不够用，所以不可避免的需要将其存到栈或者内存中，存到栈中就要记住相关的顺序， 存到内存中就要注意ds和es寄存器的段基址正确与否。没有大局观就会出现各种bug | | | | | |
| 1. 各人实验贡献与体会（每人各自撰写） | | | | | |
| 分工 : 本次实验所有内容完全由鄢锦琪完成  体会 : 本次实验中进一步加深了对段页式的理解，虽然在上学期已经考核过多次，但是在动手实现Linear2Physical，alloc\_page，free\_page时，仍然还是遇到了很多细节问题，比如在进行地址映射时，就出现了，右移22位后忘记左移4位，忘记在移位前后进行与或操作等。就会出现各种奇怪的问题。并且总是闭门造车，所以有时会花费非常多的时间用于debug而自己无法发现需要外界帮助。下次实验要更加注意细节等等。  这次实验中更感觉到一些操作系统中的精妙之处，在进行深入了解细节之前，对页的大小是4k，页目录，页表并没有什么感觉，直到在这次实验中发现，页大小是4k正好是12位2进制所能表达的最大大小，然后通过夜管理，页表和页目录来减少页的空间的使用，  觉得非常的精妙。  这次实验对细节重要性有了更深的理解，在编写相关程序时，移位的多少，与或的位数，寄存器的使用等等，都需要在敲代码时注意，不能滥竽充数，囫囵吞枣，不然就会出现很多bug，浪费很多时间。 | | | | | |
| 1. 教师评语 | | | | | |
|  | | | | | |
| 评分 | | | | | |
| 姓名 | | 学号 | | 分数 | |
|  | |  | |  | |
|  | |  | |  | |
|  | |  | |  | |
|  | |  | |  | |
| 教师签名：  年 月 日 | | | | | |