|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | |  |
| 实验名称 |  | | 实验周次 | |  |
| 姓名 | 学号 | | 专业 | | 班级 |
| 王卓 | 2021302191791 | | 网络空间安全 | | 9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） | | | | | |
|  | | | | | |
| 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） | | | | | |
| **（一）**   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。   1. **调试代码，掌握分页机制基本方法与思路**   - pmtest6.asm代码中，在212行代码前设置断点调试，分析几个循环在干什么？  为了实现调试功能，我们在bochsrc中添加magic\_break: enabled=1  20231017100628  在第212行代码之前设置断点进行调试：  20231023152009  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   20231023152232   1. 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   20231023153815   1. 查看寄存器的值   20231023153935  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   1. 继续运行到loop .1处，完成页目录表的初始化   2023102315463620231023154816  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   1. 继续单步执行，完成页表的初始化   2023102315584720231023155952  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1024^2次，完成所有页表中PTE的初始化  2023102316164520231023161902  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   1. 继续单步执行，启动分页机制   2023102316232920231023162529  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。   1. **PDE&PTE计算方法**   PDE（4KB页表）PTE（4KB页）  映射图（略）  cr3 中读取到页目录表的基地址, 然后结合线性地址的前10位获得页表的基地址, 结合中间10位获得物理页的基地址, 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。  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）？  因为每一个PDE映射1024个PTE，这些PTE的基址就存在于他们所对应的PDE中，即初始化的PageTblBase，所以PTE初始化的时候就不需要添加基地址了（通过映射图可以看出）   1. **熟悉如何获取当前系统内存布局的方法**   20231023164826  IMG_256  分析运行结果可以知道：操作系统所能使用的最大内存地址为01FE FFFFh，所以此机器拥有接近32MB的内存。  （感觉就是抄书。。。）   1. **内存地址映射关系的切换**   设置断点：  IMG_256  20231023165637  20231023170333   1. **基础题** 2. 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。 3. 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，如果超出了有什么解决办法   补充的allco\_pages和free\_pages函数见后。  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，将部分数据存储再磁盘上，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。  **（二）**   1. 分页和分段有什么区别？本次实验中，段页机制是则怎么搭配工作的？   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。   1. PDE、PTE是什么？例程中是怎么初始化的？CPU是怎么访问到PDE、PTE，从而计算出物理地址的？ 2. 开启分页机制后，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？ 3. 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，页目录与页表内存中相邻。   1. 怎么读取本机的实际物理内存信息？ 2. 如何进行地址映射与切换？   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   1. 如何实现alloc\_pages ， free\_pages  * **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：   1. 寻找可用的物理页，计算并返回物理地址    1. 使用位图来查找可用的物理页 2. \_BitMap: times 32 db 0xff ;low 1 MB is occupied(0 ~ 0xfffff) 3. times 32 db 0x0 ; 1 MB is available(0x100000 ~ 0x1fffff) 4. BitMap   equ \_BitMap  - $$ 5. BitMapLen  equ $ - \_BitMap   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。   * 1. 编写alloc\_a\_4k\_page函数计算并返回物理地址  1. alloc\_a\_4k\_page:                        ; arg none 2. ; return eax: physical address 3. ; physical address begin at 0x00000000 4. ; save reg 5. push  ds 6. push  es 7. ; fuction implement 8. xor eax, eax 9. mov ax, SelectorFlatRW 10. mov es, ax 11. mov ax, SelectorData 12. mov ds, ax 14. .search: 15. bts  [BitMap], eax 16. jnc  .find 17. inc  eax 18. cmp  eax,BitMapLen\*8 19. jl   .search 20. ; no available physical space 21. ; we should move one page back to disk 22. ; but for simplicity, we hlt 23. hlt 24. .find: 25. shl  eax,12 ;ecx -> avaiable physical addr 26. pop  es 27. pop  ds 28. 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。   1. 完成虚拟地址对物理地址的映射关系    1. 基于当前PDE和PTE找到当前页表    2. 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：   1. alloc\_pages:                               ; arg: eax : page number 2. ; return ebx : linear address 3. ; save registers 4. push ds 5. push es 6. ; function implement 7. mov bx, SelectorFlatRW 8. mov ds, bx 9. mov bx, SelectorData 10. mov es, bx 12. mov ecx, eax              ; ecx means the number of page 13. mov ebx, 4096 14. mul ebx     ; ebx means the size of pages 16. mov ebx, [es:AvaLinearAddress] ; ebx means the return value 17. add [es:AvaLinearAddress],eax  ; update the addresss of free linear address 18. push ebx                  ; save the return value 19. mov  eax, ebx 20. mov  ebx, cr3    ; assume linear2Physical(cr3) = cr3 21. and  ebx, 0xfffff000 22. and  eax, 0xffc00000 23. shr  eax, 20 24. add  ebx, eax             ; ebx means the pde item 25. mov  edx, ebx    ; edx -> missing pde address 26. mov  ebx, [ebx]           ; ebx means the corresponding page table item 27. ;    test p flag 28. test ebx, 0x0000\_0001 29. jnz  .pde\_exist 30. ;    page table is not exist 31. mov ebx, cr3 32. mov ebx, [ebx]            ; first page table address 33. and ebx, 0xfffff000 34. shl eax, 10               ; eax means the size of used pages 35. add ebx, eax 36. or  ebx, 0x0000\_0007 37. mov [edx], ebx     1. 使用alloc\_a\_4k\_page函数分配单个4KB的页，并设置对应页表项     2. 循环执行前面步骤，完成所有页的分配   具体代码如下：   1. .pde\_exist: 2. mov eax,[esp] 3. and ebx, 0xfffff000 4. and eax, 0x003ff000 5. shr eax, 10 6. add ebx, eax 7. .change\_pte: 8. call alloc\_a\_4k\_page 9. or eax, 0x00000007 10. mov [ebx] , eax 11. add ebx, 4 12. loop  .change\_pte 14. pop  ebx ;set return value 15. pop es 16. pop ds 17. ret   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。   * **实现free\_pages**   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：   1. free\_pages:              ; arg  eax,linear address , ebx  page number 2. push ds 3. push es 4. push ebx         ; save eax and ebx 5. push eax 6. mov bx, SelectorFlatRW 7. mov ds, bx 8. mov bx, SelectorData 9. mov es, bx       ; normal init 11. ; find the pde and pte 12. mov ebx, cr3 13. and ebx, 0xfffff000 14. and eax, 0xffc00000 15. shr eax, 20      ; 20 = 22 -2 16. add ebx, eax     ; ebx now means the pde item 17. mov edx, [ebx] 18. and edx, 0xfffffff8 19. mov [ebx], edx   ; set the final 3-bit zero and store back 21. mov ebx, [ebx]   ; now ebx means the first pte item 23. mov eax, [esp]          ; now eax is the liner address 24. add esp, 4 25. and ebx,0xfffff000 26. and eax,0x003ff000 27. shr eax,10 28. add ebx, eax       ;  now ebx means the right pte item 29. mov ecx, [esp]          ; here ecx means page number 30. add esp,4 31. .change\_pte:                 ; set every item's last 3-bit zero 32. mov eax, [ebx] 33. and eax, 0xfffffff8 34. mov edx, eax       ; now eax is the physical address 35. shr edx, 12 36. btr [BitMap], edx 37. mov [ebx], eax 38. add ebx,32 39. loop .change\_pte 40. pop es 41. pop ds 42. 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。   * **编写测试函数查看结果**   测试函数具体代码如下：   1. TestAllocAndFree: 2. xchg bx,bx 3. mov eax,4 4. call alloc\_pages 5. xchg bx,bx 6. mov eax,ebx 7. mov ebx,4 8. call free\_pages 9. xchg bx,bx 10. ret   测试结果如下所示：   1. cr3=0x000000200000情况下：    1. 查看alloc\_pages之前地址的映射关系   20231028202241   * 1. 查看alloc\_pages之后地址的映射关系   20231028202252   * 1. 查看alloc\_pages之后地址的映射关系   20231028202300   1. cr3=0x000000210000情况下：    1. 查看alloc\_pages之前地址的映射关系   20231028202626   * 1. 查看alloc\_pages之后地址的映射关系   20231028202636   * 1. 查看alloc\_pages之后地址的映射关系   20231028202642  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。 | | | | | |
| 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. 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） | | | | | |
| 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。 | | | | | |
| 1. 各人实验贡献与体会（每人各自撰写） | | | | | |
| 同学：王卓  此次实验为本人独立完成大部分实验内容，并主要针对**（一）部分的1、2题、基础题2以及（二）部分的1、6题**进行对应部分内容的实验报告的撰写。这次实验给我的体会如下：  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！ | | | | | |
| 1. 教师评语 | | | | | |
|  | | | | | |
| **教师评分（请填写好姓名、学号）** | | | | | |
| 姓名 | | 学号 | | 分数 | |
|  | |  | |  | |
|  | |  | |  | |
|  | |  | |  | |
|  | |  | |  | |
| 教师签名：  年 月 日 | | | | | |