**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **组 别** |  | **姓 名** | 尹浩男 | **同组实验者** |  |
| **实验项目名称** | Linux内存管理 | | | **实验日期** | 11月30日 |
| **教师评语:** |  | | | | |
| **实验成绩：** | | | **指导教师（签名）：**  2020年 月 日 | | |
| **一．实验目的**  1. 了解Linux物理内存管理方法  2. 了解IA32架构中的逻辑地址映射机制，包括分段和分页  3. 了解Linux的分段机制及其特色  4. 理解并掌握Linux从线性地址到物理地址的映射机制  5. 理解Linux进程虚拟地址空间机制  6. 理解Linux虚拟内存区域vma的管理方法  **二．实验内容**  计算vma每个vma区域的大小，以及代码段和数据段的大小，并通过测试程序显示。  **三、实验步骤**   |  | | --- | | 第一步make编译  root@yhnComputer:/yhnCode/vma# make  make -C /lib/modules/5.4.0-58-generic/build M=/yhnCode/vma modules  make[1]: Entering directory '/usr/src/linux-headers-5.4.0-58-generic'    CC [M]  /yhnCode/vma/vma.o    Building modules, stage 2.    MODPOST 1 modules    CC [M]  /yhnCode/vma/vma.mod.o    LD [M]  /yhnCode/vma/vma.ko  make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-58-generic'  rm -r -f .tmp\_versions \*.mod.c .\*.cmd \*.o \*.symvers  第二步 安装模块  root@yhnComputer:/yhnCode/vma# insmod vma.ko  第三步 编译测试程序  root@yhnComputer:/yhnCode/vma# gcc vma\_test.c -o test  第四步 执行test：  root@yhnComputer:/yhnCode/vma# ./test |   **四. 实验结果**   |  | | --- | | This is parent pid 31041  This is child pid 31042  Info from the Memory Management structure for task 'test' (pid=31042)  pgd=FFFF8FA66BA8A000 mmap=FFFF8FA58B8E9040 map\_count=22 mm\_users=1 mm\_count=1  start\_code=556805074000 end\_code=556805075545  start\_data=556805077D60 end\_data=556805078010  start\_brk=556805DAC000 brk=556805DCD000  arg\_start=7FFE073F81A1 arg\_end=7FFE073F81A8  env\_start=7FFE073F81A8 env\_end=7FFE073F8FF1  start\_stack=7FFE073F6780 down\_to=7FFE073D5780 <--- stack grows downward  List of the Virtual Memory Areas for task 'test' (pid=31042)  1 vm\_start=556805074000 vm\_end=556805075000 r--p  2 vm\_start=556805075000 vm\_end=556805076000 r-xp  3 vm\_start=556805076000 vm\_end=556805077000 r--p  4 vm\_start=556805077000 vm\_end=556805078000 r--p  5 vm\_start=556805078000 vm\_end=556805079000 rw-p  6 vm\_start=556805DAC000 vm\_end=556805DCD000 rw-p  7 vm\_start=7FF956826000 vm\_end=7FF95684B000 r--p  8 vm\_start=7FF95684B000 vm\_end=7FF9569C3000 r-xp  9 vm\_start=7FF9569C3000 vm\_e  Info from the Memory Management structure for task 'test' (pid=31041)  pgd=FFFF8FA648FB0000 mmap=FFFF8FA5C75D6A90 map\_count=22 mm\_users=1 mm\_count=1  start\_code=556805074000 end\_code=556805075545  start\_data=556805077D60 end\_data=556805078010  start\_brk=556805DAC000 brk=556805DCD000  arg\_start=7FFE073F81A1 arg\_end=7FFE073F81A8  env\_start=7FFE073F81A8 env\_end=7FFE073F8FF1  start\_stack=7FFE073F6780 down\_to=7FFE073D5780 <--- stack grows downward  List of the Virtual Memory Areas for task 'test' (pid=31041)  1 vm\_start=556805074000 vm\_end=556805075000 r--p  2 vm\_start=556805075000 vm\_end=556805076000 r-xp  3 vm\_start=556805076000 vm\_end=556805077000 r--p  4 vm\_start=556805077000 vm\_end=556805078000 r--p  5 vm\_start=556805078000 vm\_end=556805079000 rw-p  6 vm\_start=556805DAC000 vm\_end=556805DCD000 rw-p  7 vm\_start=7FF956826000 vm\_end=7FF95684B000 r--p  8 vm\_start=7FF95684B000 vm\_end=7FF9569C3000 r-xp  9 vm\_start=7FF9569C3000 vm\_end=7FF956A0D000 r--p  nd=7FF956A0D000 r--p  10 vm\_start=7FF956A0D000 vm\_end=7FF956A0E000 ---p  10 vm\_start=7FF956A0D000 vm\_end=7FF956A0E000 ---p  11 vm\_start=7FF956A0E000 vm\_end=7FF956A11000 r--p  11 vm\_start=7FF956A0E000 vm\_end=7FF956A11000 r--p  12 vm\_start=7FF956A11000 vm\_end=7FF956A14000 rw-p  12 vm\_start=7FF956A11000 vm\_end=7FF956A14000 rw-p  13 vm\_start=7FF956A14000 vm\_end=7FF956A1A000 rw-p  13 vm\_start=7FF956A14000 vm\_end=7FF956A1A000 rw-p  14 vm\_start=7FF956A2C000 vm\_end=7FF956A2D000 r--p  14 vm\_start=7FF956A2C000 vm\_end=7FF956A2D000 r--p  15 vm\_start=7FF956A2D000 vm\_end=7FF956A50000 r-xp  15 vm\_start=7FF956A2D000 vm\_end=7FF956A50000 r-xp  16 vm\_start=7FF956A50000 vm\_end=7FF956A58000 r--p  16 vm\_start=7FF956A50000 vm\_end=7FF956A58000 r--p  17 vm\_start=7FF956A59000 vm\_end=7FF956A5A000 r--p  17 vm\_start=7FF956A59000 vm\_end=7FF956A5A000 r--p  18 vm\_start=7FF956A5A000 vm\_end=7FF956A5B000 rw-p  18 vm\_start=7FF956A5A000 vm\_end=7FF956A5B000 rw-p  19 vm\_start=7FF956A5B000 vm\_end=7FF956A5C000 rw-p  19 vm\_start=7FF956A5B000 vm\_end=7FF956A5C000 rw-p  20 vm\_start=7FFE073D8000 vm\_end=7FFE073F9000 rw-p  20 vm\_start=7FFE073D8000 vm\_end=7FFE073F9000 rw-p  21 vm\_start=7FFE073FC000 vm\_end=7FFE073FF000 r--p  21 vm\_start=7FFE073FC000 vm\_end=7FFE073FF000 r--p  22 vm\_start=7FFE073FF000 vm\_end=7FFE07400000 r-xp  22 vm\_start=7FFE073FF000 vm\_end=7FFE07400000 r-xp  CR3=12BA8A004 mm->pgd=0000000032d32ae1 mm->map\_count=22  CR3=108FB0003 mm->pgd=00000000fd68b726 mm->map\_count=22 |   **五. 分析**  在Linux内核中，这样的区域被称之为虚拟内存区域(virtual memory areas),简称vma。一个vma就是一块连续的线性地址空间的抽象，它拥有自身的权限(可读，可写，可执行等等) ，每一个虚拟内存区域都由一个相关的struct vm\_area\_struct结构来描述，这次实验给出了vma的空间的详细情况。因为实验环境是在32位ubuntu下进行的，64位操作系统不支持汇编指令movl，我使用的版本是64位的ubuntu20，所以需要更换汇编指令movl =》 mov 寄存器从ecx更改为rcx (之后在x86正式导入64位架构后,四个通用寄存器(RAX, RBX, RCX, RDX),它们较低的32位分别与原本32位的通用寄存器(EAX, EBX, ECX, EDX)重叠共用) | | | | | |