LOU3977594477 L4 2020-12-28 22:19:22 地址映射与共享

17 0

“操作系统原理与实践”实验报告

地址映射与共享

问题 1.输入命令u/7反汇编，查看变量i对应的逻辑地址 逻辑地址找虚拟地址要通过段表，也就是IDT表，然后IDT表要根据LDTR寄存器和GDT表，对应的命令就是sreg 根据ds(代码段)寄存器查找IDT表，得到基址，然后通过基址 + 逻辑地址 = 虚拟地址 根据虚拟地址找到物理地址，核心就是查找页表。 最后找到了物理地址就需要使用命令setpmem (物理地址) 4 0来修改变量i的值，然后命令c继续运行就可以退出程序了。 2. 得到的i的物理地址可能会不同。在linux0.11中，因为有虚拟内存和段页结合的内存管理机制，get_free_page()在物理页框中找出空闲页是很随意的——一只要是空闲的页框就直接拿来用。而test.exe重启后，其各个段在上一次执行时使用的物理页框很可能已经被其他进程占用了。所以这一次其data段有可能会被分配到别的物理页框中去。所以得到的物理地址可能会不同。

1.在oslab/oslab中编写test.c文件

test.c

```
#include <stdio.h>

int i = 0x12345678;

int main(void)
{
  printf("The logical/virtual address of i is 0x%08x", &i);
  fflush(stdout);

  while (i)
    ;

  return 0;
}
```

```
shiyanlou@bc20505846e7:~/oslab/oslab$ sudo ./mount-hdc
shiyanlou@bc20505846e7:~/oslab/oslab$ cp test.c hdc/usr/root/
```

2.

```
shiyanlou@85c418492e5a:~/oslab/oslab$ ./dbg-asm
========================================================================
                    Bochs x86 Emulator 2.3.7
            Build from CVS snapshot, on June 3, 2008
========================================================================
00000000000i[     ] reading configuration from ./bochs/bochsrc.bxrc
00000000000i[     ] installing x module as the Bochs GUI
00000000000i[     ] using log file ./bochsout.txt
Next at t=0
(0) [0xfffffff0] f000:fff0 (unk. ctxt): jmp far f000:e05b         ; ea5b
e000f0
<bochs:1> c
^CNext at t=252616810
```

3.运行test.c

```
[/usr/root]# gcc -o test test.c
[/usr/root]# ./test
LQD The logical/virtual address of i is 0x00003004
```

4.使用反汇编指令u/7，显示从当前位置(while(1))开始的7条指令的反汇编指令

```
<bochs:4> u/7
10000067: (                   ): cmp dword ptr ds:0x3004, 0x00000000 ;
833d0430000000
1000006e: (                   ): jz .+0x00000004           ; 7404
10000070: (                   ): jmp .+0xfffffff5          ; ebf5
10000072: (                   ): add byte ptr ds:[eax], al ; 0000
10000074: (                   ): xor eax, eax              ; 31c0
10000076: (                   ): jmp .+0x00000000          ; eb00
10000078: (                   ): leave                     ; c9
```

实验数据

学习时间	64分钟
操作时间	11分钟
按键次数	70次
实验次数	5次
报告字数	6537字
是否完成	完成

评分

未评分

下一篇

篇

相关报告

操作系统原理与实践: 熟悉实验环境 实验报告

操作系统原理与实践: 熟悉实验环境 实验报告

操作系统原理与实践: 基于内核栈切换的进程切换 实验报告

操作系统原理与实践: 熟悉实验环境 实验报告

操作系统原理与实践: 信号量的实现和应用 实验报告

5.用sreg命令

```
<bochs:5> sreg
cs:s=0x000f, dl=0x00000002, dh=0x10c0fa00, valid=1
ds:s=0x0017, dl=0x00003fff, dh=0x10c0f300, valid=3
ss:s=0x0017, dl=0x00003fff, dh=0x10c0f300, valid=1
es:s=0x0017, dl=0x00003fff, dh=0x10c0f300, valid=1
fs:s=0x0017, dl=0x00003fff, dh=0x10c0f300, valid=1
gs:s=0x0017, dl=0x00003fff, dh=0x10c0f300, valid=1
ldtr:s=0x0068, dl=0x52d00068, dh=0x000082fd, valid=1
tr:s=0x0060, dl=0x52e80068, dh=0x00008bfd, valid=1
gdtr:base=0x00005cb8, limit=0x7ff
idtr:base=0x000054b8, limit=0x7ff
```

6.使用命令xp /32w 0x00005cb8查看GDT表

```
<bochs:6> xp /32w 0x00005cb8
[bochs]:
0x00005cb8 <bogus+       0>:    0x00000000      0x00000000      0x00000f
ff      0x00c09a00
0x00005cc8 <bogus+      16>:    0x00000fff      0x00c09300      0x000000
00      0x00000000
0x00005cd8 <bogus+      32>:    0xa4280068      0x00008901      0xa41000
68      0x00008201
0x00005ce8 <bogus+      48>:    0xf2e80068      0x000089ff      0xf2d000
68      0x000082ff
0x00005cf8 <bogus+      64>:    0xd2e80068      0x000089ff      0xd2d000
68      0x000082ff
0x00005d08 <bogus+      80>:    0x12e80068      0x000089fc      0x12d000
68      0x000082fc
0x00005d18 <bogus+      96>:    0x52e80068      0x00008bfd      0x52d000
68      0x000082fd
0x00005d28 <bogus+     112>:    0xe2e80068      0x000089f8      0xe2d000
68      0x000082f8
```

7.GDT表中的每一项占64位(8个字节)，所以要查找的项的地址是0x00005cb8 + 13*8*，使用命令:*xp /2w 0x00005cb8 + 13*8

```
<bochs:8> xp /8w 0x00fd52d0
[bochs]:
0x00fd52d0 <bogus+       0>:    0x00000000      0x00000000      0x000000
02      0x10c0fa00
0x00fd52e0 <bogus+      16>:    0x00003fff      0x10c0f300      0x000000
00      0x00fd6000
```

8.再次输入sreg

```
<bochs:9> sreg
cs:s=0x000f, dl=0x00000002, dh=0x10c0fa00, valid=1
ds:s=0x0017, dl=0x00003fff, dh=0x10c0f300, valid=3
ss:s=0x0017, dl=0x00003fff, dh=0x10c0f300, valid=1
es:s=0x0017, dl=0x00003fff, dh=0x10c0f300, valid=1
fs:s=0x0017, dl=0x00003fff, dh=0x10c0f300, valid=1
gs:s=0x0017, dl=0x00003fff, dh=0x10c0f300, valid=1
ldtr:s=0x0068, dl=0x52d00068, dh=0x000082fd, valid=1
tr:s=0x0060, dl=0x52e80068, dh=0x00008bfd, valid=1
gdtr:base=0x00005cb8, limit=0x7ff
idtr:base=0x000054b8, limit=0x7ff
```

9.用命令验证线性地址是不是对的:calc ds:0x3004

```
<bochs:10> calc ds:0x3004
0x10003004 268447748
```

10.在IA-32(英特尔的32位CPU架构)下，页目录表的位置由CR3寄存器指引，用creg命令可以看到

```
<bochs:11> creg
CR0=0x8000001b: PG cd nw ac wp ne ET TS em MP PE
CR2=page fault laddr=0x10002fb0
CR3=0x00000000
    PCD=page-level cache disable=0
    PWT=page-level writes transparent=0
CR4=0x00000000: osxmmexcpt osfxsr pce pge mce pae pse de tsd pvi vme
```

11.输入xp /68w 0

```
07      0x00004027
0x00000010 <bogus+      16>:    0x00000000      0x0002a1d0      0x
00      0x00000000
0x00000020 <bogus+      32>:    0x00000000      0x00000000      0x
00      0x00000000
0x00000030 <bogus+      48>:    0x00000000      0x00000000      0x
00      0x00000000
0x00000040 <bogus+      64>:    0x00ffe027      0x00000000      0x
00      0x00000000
0x00000050 <bogus+      80>:    0x00000000      0x00000000      0x
00      0x00000000
0x00000060 <bogus+      96>:    0x00000000      0x00000000      0x
00      0x00000000
0x00000070 <bogus+     112>:    0x00000000      0x00000000      0x
00      0x00000000
0x00000080 <bogus+     128>:    0x00ff3027      0x00000000      0x
00      0x00000000
0x00000090 <bogus+     144>:    0x00000000      0x00000000      0x
00      0x00000000
0x000000a0 <bogus+     160>:    0x00000000      0x00000000      0x
00      0x00000000
0x000000b0 <bogus+     176>:    0x00000000      0x00000000      0x
00      0x00ffb027
0x000000c0 <bogus+     192>:    0x00ff6027      0x00000000      0x
00      0x00000000
0x000000d0 <bogus+     208>:    0x00000000      0x00000000      0x
00      0x00000000
0x000000e0 <bogus+     224>:    0x00000000      0x00000000      0x
00      0x00000000
0x000000f0 <bogus+     240>:    0x00000000      0x00000000      0x
00      0x00ffa027
0x00000100 <bogus+     256>:    0x00fa6027      0x00000000      0x
```

12.用命令查看: xp /w 0 + 64*4* 在这里插入图片描述 其中的*027*是属性，显然*P=1*，这里还也可以分析出其它的属性。 页表所在的物理页框号为*0x00fa6*，即页表在物理内存为*0x00fa6000*处，从该位置开始查找*3*号页表项*(*每个页表项*4*个字节*)*，用命令*: xp /w 0x00fa6000 + 3*4

```
<bochs:13> xp /w 0 + 64*4
[bochs]:
0x00000100 <bogus+       0>:    0x00fa6027
<bochs:14> xp /w 0x00fa6000 + 3*4
[bochs]:
0x00fa600c <bogus+       0>:    0x00fa3067
```

13.用命令: xp /w 0x00fa3004:

```
<bochs:15> xp /w 0x00fa3004
[bochs]:
0x00fa3004 <bogus+       0>:    0x12345678
```

14.setpmem 0x00fa3004 4 0，直接修改物理地址的值，使得变量i为0

```
<bochs:16> setpmem 0x00fa3004 4 0
```

producer.c #include <unistd.h> #include <syscall.h> #include <sys/shm.h> #include <semaphore.h> #include <fcntl.h> #include <stdio.h>

define BUF_SIZE 10

define COUNT 500

define KEY 183

define SHM_SIZE (BUF_SIZE+1)*sizeof(short)

int main(int argc,char ** argv) { int pid;//该进程的id unsigned short count = 0;//生产资源的个数 int shm_id;//共享物理内存空间的id short *shmp;//操作共享内存的逻辑地址 sem_t *empty;//三个实现进程间的同步 sem_t *full; sem_t *mutex;

```
//关闭原来的信号量
sem_unlink("empty");
sum_unlink("full");
sum_unlink("mutex");

//新打开三个信号量
empty = sem_open("empty",0_CREAT|O_EXCL,0666,10);
full = sem_open("full",0_CREAT|O_EXCL,0666,0);
mutex = sem_open("mutex",0_CREAT|O_EXCL,0666,1);
if(empty == SEM_FAILED || full == SEM_FAILED || mutex == SEM_FAILED)
{
    //申请信号量失败
    printf("sem_open error!\n");
    return -1;
}

//使用KEY值申请一块共享物理内存
shm_id = shmget(KEY,SHM_SIZE,IPC_CREAT|0666);
if(shm_id == -1)
{
    //申请共享内存失败
    printf("shmget error!\n");
    return -1;
}
shmp = (short*)shmat(shm_id,NULL,0);//返回共享物理内存的逻辑地址


pid = syscall(SYS_getpid);//得到进程的pid

//生产者生产出资源
while(count <= COUNT)
{
    sem_wait(empty);//P(empty)
    sem_wait(mutex);//P(mutex)

    printf("Producer 1 process %d : %d\n",pid,count);
    fflush(stdout);
    *(shmp++) = count++;
    if(!(count % BUF_SIZE))
    {
        shmp -= 10;
    }

    sem_post(mutex);//V(mutex)
    sem_post(full);//V(full)
}
return 0;
```

} consumer.c #include <unistd.h> #include <syscall.h> #include <sys/shm.h> #include <semaphore.h> #include <fcntl.h> #include <stdio.h>

define BUF_SIZE 10

define KEY 183

int main() { int pid; int shm_id; short *shmp; short *index; sem_t *empty; sem_t *full; sem_t *mutex;

```
shm_id = shmget(KEY,0,0);//使用和生产者同一个KEY值，会返回同一个shm_id(指向同一个内存空间)
if(shm_id == -1)
{
    //申请共享内存失败
    printf("shmget error!\n");
    return -1;
}
shmp = (short*)shmat(shm_id,NULL,0);//返回共享物理内存的逻辑地址
index = shmp + BUF_SIZE;
*index = 0;

//打开生产者那里创建的三个信号量
empty = sem_open("empty",0);
full = sem_open("full",0);
mutex = sem_open("mutex",0);
if(empty == SEM_FAILED || full == SEM_FAILED || mutex == SEM_FAILED)
{
    //申请信号量失败
    printf("sem_open error!\n");
    return -1;
}

if(!sysvall(SYS_fork))
{
    pid = syscall(SYS_getpid);//得到进程的pid

    //消费者1开始消费资源
    while(1)
    {
        sem_wait(full);//P(full)
        sem_wait(mutex);//P(mutex)

        printf("Consumer 1 process %d : %d\n",pid,shem[*index]);
        fflush(stdout);
        if(*index == 9)
        {
            *index = 0;
        }
        else
        {
            (*index)++;
        }

        sem_post(mutex);//V(mutex)
        sem_post(empty);//V(empry)
    }
    return 0;
}

if(!sysvall(SYS_fork))
{
    pid = syscall(SYS_getpid);//得到进程的pid

    //消费者2开始消费资源
    while(1)
    {
        sem_wait(full);
        sem_wait(mutex);

        printf("Consumer 2 process %d : %d\n",pid,shem[*index]);
        fflush(stdout);
        if(*index == 9)
        {
            *index = 0;
        }
        else
        {
            (*index)++;
        }

        sem_post(mutex);
        sem_post(empty);
    }
    return 0;
}

if(!sysvall(SYS_fork))
{
    pid = syscall(SYS_getpid);//得到进程的pid

    //消费者3开始消费资源
    while(1)
```

```
    {
        sem_wait(full);
        sem_wait(mutex);

        printf("Consumer 3 process %d : %d\n",pid,shem[*index]);
        fflush(stdout);
        if(*index == 9)
        {
            *index = 0;
        }
        else
        {
            (*index)++;
        }

        sem_post(mutex);
        sem_post(empty);
    }
    return 0;
}
return 0;
```

}










Markdown 语法

请输入想说的话

0 / 2000

发表评论

最新评论

/