

Linux 内核在 x86_64 CPU 中地址映射

<http://www.ilinuxkernel.com>

在《Linux内存地址映射》(<http://linuxkernel.com/?p=1276>)，详细介绍了在32位x86 CPU中Linux内核地址映射过程，并且给出实验验证整个地址映射过程。

64位CPU中，地址映射稍微复杂，本文介绍Linux内核在x86_64 CPU中地址映射过程，同样给出实验和源码，验证整个地址映射过程。

1 x86_64 CPU中逻辑地址（段式）映射

x86_64段式地址过程和x86一致，即各段起始地址都是0，区别在于段大小不再是4G。在实验过程中，我们再来看每段大小的限制，不再理论分析。

2 x86_64 CPU中线性地址（页式）映射

本文只考虑最常使用4K页面时的线性地址映射，图1是4K页面时，线性地址映射模型。

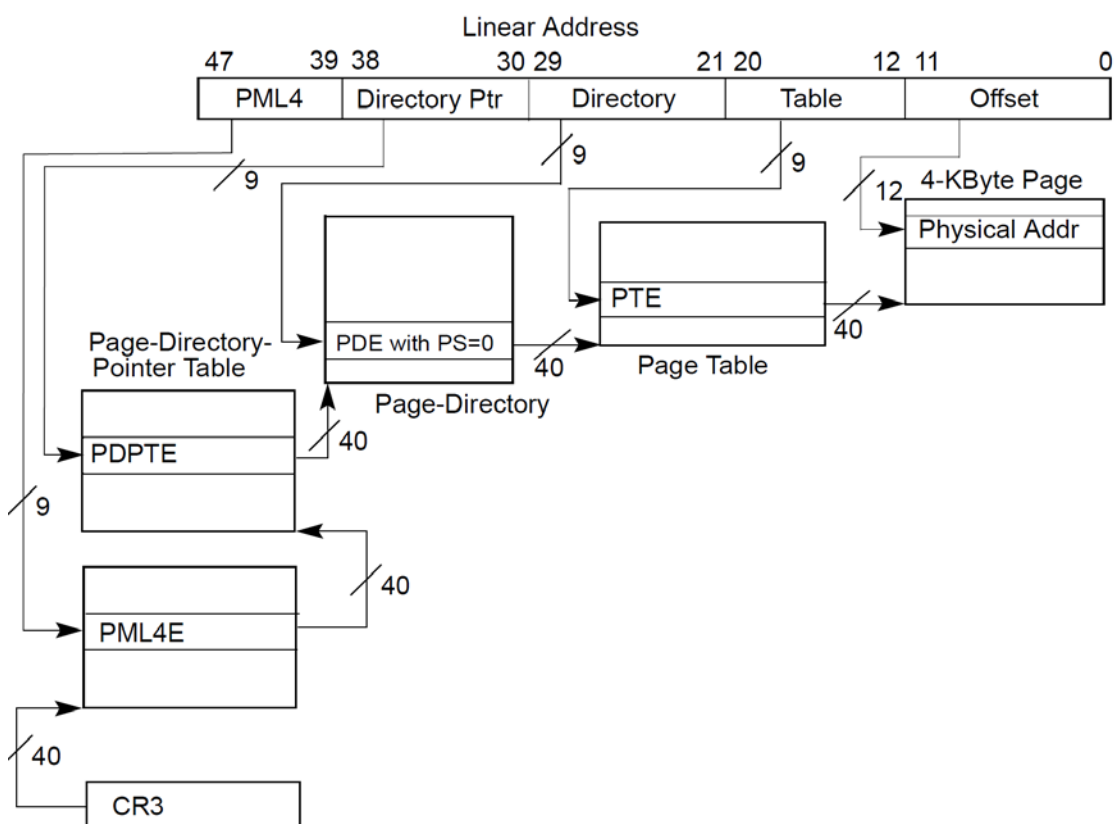


图1 IA32-e模式下4K页面线性地址映射

在x86_64 CPU架构中，页面线性地址映射称为IA32-e模式。

简单概括一下上面线性地址映射内容：

(1) 线性地址是48bit

注意x86_64线性地址不是64bit，物理地址也不是64位，Intel当前CPU最高物理地址是52bit，但实际支持的物理内存地址总线宽度是40bit，见图2。

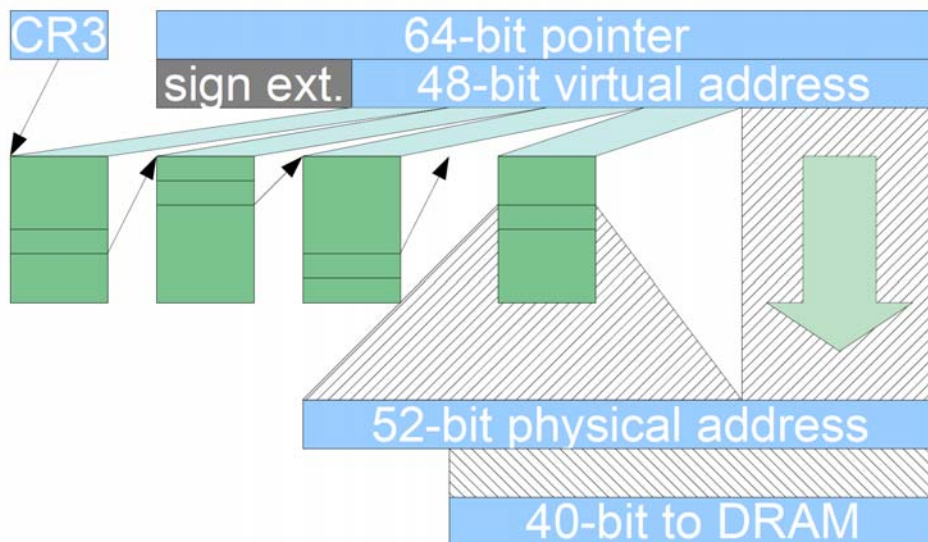


图2 IA32-e模式下线性地址映射

(2) 页面映射分为4级

48bit线性地址分为5段、bit位宽度分别为9、9、9、9、12。映射方法和x86一致，就是一层层查表。

(3) CR3寄存器保存最高一级表的起始物理地址

(4) 每个表项的大小都为8字节

3 Linux x86_64地址映射实验

实验的必要条件（如读取CR3和GDTR寄存器、访问实际物理内存）和相关代码，已在《Linux内存地址映射》（<http://ilinuxkernel.com/?p=1276>）中详细描述。若不了解，请先阅读这篇文章。

实验环境为：RHEL6，内核版本为2.6.32-220.el6.x86_64。物理机器为：华为RH2285

机架服务器，2xE5620，80GB RAM。

我们仅对x86下的实验代码稍作修改，就可以在x86_64环境下运行。下面是一个运行实例。

```
unsigned long tmp;
tmp = 0x12345678beaf5dde;
static int cr_fd = - 1;

printf("tmp address:0x%lX\n", &tmp);
FILE_TO_BUF(REGISTERINFO, cr_fd);

printf("%s", buf);

while(1);
```

```
[root@RH2285 Memory_Address_Mapping_x86-64]# insmod sys_reg.ko
[root@RH2285 Memory_Address_Mapping_x86-64]# insmod dram.ko
[root@RH2285 Memory_Address_Mapping_x86-64]# mknod /dev/sys_reg c 85 0
[root@RH2285 Memory_Address_Mapping_x86-64]# ./mem_map
tmp address:0x7FFF3FFD8FE8
cr4=6E0 PSE=0 PAE=1
cr3=62E604000 cr0=80050033
bgd:0xFFFF88062E604000
gdtr address:FFFF8800282A4000, limit:7F
```

应用程序中，tmp变量的临时地址为0x7FFF3FFD8FE8。我们就针对这个线性地址，逐步映射到实际物理地址，看这个地址的数据是否真的为“0x12345678beaf5dde”。

注意：在编译fileview可执行文件时，要加上-D_LARGEFILE64_SOURCE参数，否则难以识别4G以上的物理内存。

```
[root@RH2285 Memory_Address_Mapping_x86-64]# gcc -o fileview
-D_LARGEFILE64_SOURCE fileview.c
```

应用程序中，tmp变量的临时地址为0x7FFF3FFD8FE8。我们就针对这个线性地址，逐步映射到实际物理地址，看这个地址的数据是否真的为“0x12345678beaf5dde”。

3.1 段式地址映射过程

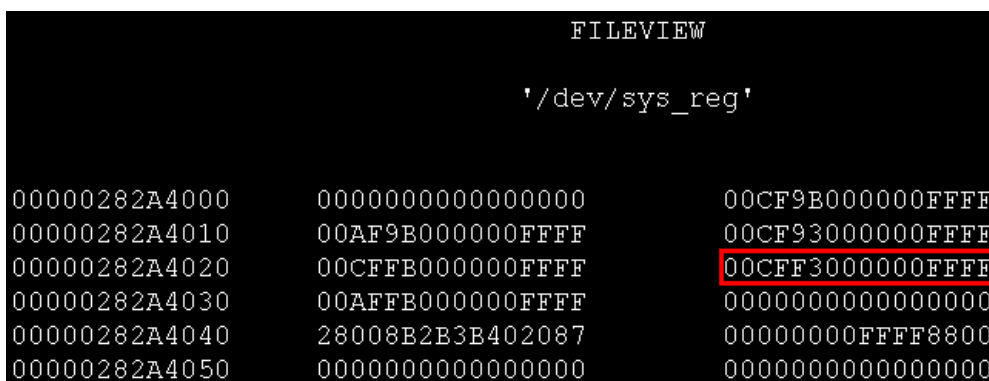
我们程序读取到的GDTR寄存器值为0xFFFF8800282A4000，对应的物理地址为

0x282A4000（内核地址仅是减去一个偏移量，这里不做详细解释）。

在x86_64内核中，用户态的DS段寄存器设置为43，对应的值为

```
#define GDT_ENTRY_DEFAULT_USER32_CS 4
#define GDT_ENTRY_DEFAULT_USER_DS 5
#define GDT_ENTRY_DEFAULT_USER_CS 6
#define __USER32_CS (GDT_ENTRY_DEFAULT_USER32_CS * 8 + 3)
#define __USER32_DS __USER_DS

[GDT_ENTRY_KERNEL32_CS] = GDT_ENTRY_INIT(0xc09b, 0, 0xffffffff),
[GDT_ENTRY_KERNEL_CS] = GDT_ENTRY_INIT(0xa09b, 0, 0xffffffff),
[GDT_ENTRY_KERNEL_DS] = GDT_ENTRY_INIT(0xc093, 0, 0xffffffff),
[GDT_ENTRY_DEFAULT_USER32_CS] = GDT_ENTRY_INIT(0xc0fb, 0, 0xffffffff),
[GDT_ENTRY_DEFAULT_USER_DS] = GDT_ENTRY_INIT(0xc0f3, 0, 0xffffffff),
[GDT_ENTRY_DEFAULT_USER_CS] = GDT_ENTRY_INIT(0xa0fb, 0, 0xffffffff),
```



```
FILEVIEW
'/dev/sys_reg'
00000282A4000 0000000000000000 00CF9B000000FFFF
00000282A4010 00AF9B000000FFFF 00CF93000000FFFF
00000282A4020 00CFFB000000FFFF 00CFF3000000FFFF
00000282A4030 00AFFB000000FFFF 0000000000000000
00000282A4040 28008B2B3B402087 00000000FFFF8800
00000282A4050 0000000000000000 0000000000000000
```

对照段式映射过程，段式映射的结果是逻辑地址和线性地址是一样的，即线性地址也是 0x 7FFF3FFD8FE8。

3.2 页式地址映射过程

为了方便观察线性地址映射过程，我们先将地址0x7FFF3FFD8FE8使用二进制，并分成5段。

```
11111111 111111100 111111111 111011000 111111101000
```

3.2.1 第一级映射

CR3寄存器的值为0x62E604000，这是第一级映射表的起始物理地址。这张表中保存着第二级映射表的物理地址。

$0x62E604000 + 11111111b * 8 = 0x62E6047F8$ 。这个物理单元保存的就是第二级映射表起始地址。

```

FILEVIEW
'/dev/sys_reg'

000062E6047F0    000000061FA56067    000000062C42C067
000062E604800    0000000000000000    0000000000000000
000062E604810    0000000000000000    0000000000000000
000062E604820    0000000000000000    0000000000000000

```

第二级映射表的起始地址为：0x62C42C000（067后面12bit是页面属性）。

3.2.2 第二级映射

第二级映射的任务是找到第三级映射表的起始地址。

$0x62C42C000 + 111111100b * 8 = 0x62C42CFE0$

0x62C42CFE0地址单元中的数据，就是第三级映射表的物理地址。

```

FILEVIEW
'/dev/sys_reg'

000062C42CFE0    000000060D09F067    0000000000000000
000062C42CFF0    0000000000000000    0000000000000000
000062C42D000    FFFF88062DDA9000    FFFF88062DF89000
000062C42D010    0000000000000000    FFFF88062C42D0C0

```

第三级映射表的起始地址为：0x60D09F000（067后面12bit是页面属性）。

3.2.3 第三级映射

第三级映射的任务是找到第四级映射表的起始地址。

$0x60D09F000 + 111111111b * 8 = 0x60D09FFF8$

0x60D09FFF8地址单元中的数据，就是第四级映射表的物理地址。

```

FILEVIEW

'/dev/sys_reg'

000060D09FFF0      0000000000000000      000000062EB1067
000060D0A0000      656C732F6E69622F      007065656C007065
000060D0A0010      0000000000000000      0000000000000000
000060D0A0020      0000000000000000      0000000000000000

```

第四级映射表的起始地址为：0x62EB81000（067后面12bit是页面属性）。

3.2.4 第四级映射

第四级映射的任务是找到临时变量tmp所在的物理页面起始地址。

$0x62EB81000 + 111011000b * 8 = 0x62EB81EC0$ 。

0x62EB81EC0地址单元中的数据，就是物理页面起始地址。

```

FILEVIEW

'/dev/sys_reg'

000062EB81EC0      80000006016E3067      800000061D603067
000062EB81ED0      8000000601BD7067      0000000000000000
000062EB81EE0      0000000000000000      0000000000000000
000062EB81EF0      0000000000000000      0000000000000000

```

经过这级页面映射，我们终于得到了tmp变量所在内存页面物理地址为：0x6016E3000（067后面12bit是页面属性）。

3.2.5 最终物理地址计算

tmp变量所在内存页面物理地址为：0x6016E3000，这个地址仅是物理页面地址，但tmp变量仅占8字节。tmp变量所在的物理地址为：

$0x6016E3000 + 11111101000b = 0x6016E3FE8$ 。

经过4级页面映射，我们找到了tmp变量的临时地址为0x7FFF3FFD8FE8对应的实际物理地址为0x6016E3FE8。现在来看看0x6016E3FE8地址单元中存放的实际数据。

```
FILEVIEW
'/dev/sys_reg'
00006016E3FE0    00007FFF3FFD90D0    12345678BEAF5DDE
00006016E3FF0    0000000000000000    00000038F6C1EC5D
00006016E4000    663778302026205D    73614D202F2F203B
00006016E4010    61622074756F206B    6574617220636973
```

没错，0x6016E3FE8地址单元中存放的实际数据确实为“0x12345678beaf5dde”！

至此，我们完整地验证了x86_64机制下的Linux地址映射过程。

实验源码下载地址：

http://www.ilinuxkernel.com/files/Memory_Address_Mapping_x86-64.tar.bz2