ICS 第九章

【虚拟内存机制】

1．在某一 64 位体系结构中，每页的大小为 4KB，采用的是三级页表，每张页表占据 1 页，页表项长度为 8 字节。则虚拟地址的位数为_____Bit。如果要映射满 64 位的虚拟地址空间，可通过增加页表级数来解决，那么至少要增加到_____级页表。这个体系结构支持多种页大小，最小的三个页大小分别是 4KB、_____MB、_____GB

【答】VPO 长度为 12 位。4KB/32=512 个条目每页表。因此 VPN 长度均为 9 位，虚拟地址长度为 12+9+9+9=39 位。由于 12+9*5=57<64，12+9*6=66>=64，因此需要采用六级页表才能映射满。　2　1

2．Intel IA32 体系中，每页的大小为 4KB，采用的是二级页表，每张页表占据一页，每个页表项（PTE、PDE）的长度均为 4 字节。支持的物理地址空间为 36 位。

如果采用二级翻译，那么每个 PDE 条目格式如下：（物理地址 35~32 位必定为 0）

	31~12	11	10	9	8	7	6	5	4	3	2	1	0
PDE（4KB 页）	指向的页表的物理地址 31~12 位					0					US	RW	V

每个 PTE 条目格式如下：

	31~12	11	10	9	8	7	6	5	4	3	2	1	0
PTE（4KB 页）	虚拟页的物理地址 31~12 位					0					US	RW	V

如果采用一级翻译（大页模式），那么每个 PDE 条目格式如下：

	31~22	21~17	16~13	12	11	10	9	8	7	6	5	4	3	2	1	0
PDE（4MB 页）	虚拟页的物理地址 31~22 位	00000	物理 35~32 位						1					US	RW	V

上表中的最低位均为第 0 位。部分位的意义如下：

V=1：当前的条目有效（指向的页在物理内存中）
RW=1：指向的区域可写。只有两级页表均为 1 的时候，该虚拟内存地址才可以写。
US=1：指向的区域用户程序可访问。只有两级页表均为 1 的时候，该虚拟内存地址才可以被用户程序访问。

某一时刻，一级页表的起始地址为 0x00C188000。部分物理内存中的数据如下：

物理地址	内容	物理地址	内容	物理地址	内容	物理地址	内容
00C188000	63	00C188001	A0	00C188002	67	00C188003	C0
00C188004	0D	00C188005	A0	00C188006	F0	00C188007	A5
00C188008	67	00C188009	A0	00C18800A	32	00C18800B	0D
00C1880C0	67	00C1880C1	30	00C1880C2	88	00C1880C3	C1
00C188300	E7	00C188301	00	00C188302	80	00C188303	9A
00C188C00	**67**	**00C1880C1**	**80**	**00C1880C2**	**18**	**00C1880C3**	**0C**
00D32A294	67	00D32A295	C0	00D32A296	83	00D32A297	67
00D32A298	C0	00D32A299	C0	00D32A29A	BB	00D32A29B	DC
00D32AA5C	67	00D32AA5D	C0	00D32AA5E	83	00D32AA5F	9A
00DA0C294	45	00DA0C295	82	00DA0C296	77	00DA0C297	67
00DA0C298	67	00DA0C299	83	00DA0C29A	29	00DA0C29B	44
00DA0CA5C	00	00DA0CA5D	9A	00DA0CA5E	88	00DA0CA5F	EF

不采用 TLB 加速翻译。

PART A. 现在需要访问虚拟内存地址 0x00A97088。

(1) 将该地址拆成 VPN1+VPN2+VPO

VPN1	VPN2	VPO
0x**2**	0x**297**	0x**088**

(2) 对应的 PDE 条目的物理地址是 0x00C188008，读出第二级页表的起始地址为 0x00D32A000，PTE 条目的起始地址为 0x00D32AA5C，因此翻译得到的物理地址为 0x09A83C088。

(3) 用户模式能否访问该地址？[Y/N] 能否写该地址？[Y/N] YY

PART B. 现在需要访问虚拟内存地址 0x3003C088。

最终翻译得到的物理地址为 0x09A83C088。

【答】VPN1=0xC0，PDE 条目物理地址为 0x00C188300，读出这一页是大页，地址高 35~22 位为 0x09A800000。因此物理地址为 0x09A83C088。

PART C. 下列 IA32 汇编代码执行结束以后，%eax 的值是多少？假设一开始%ebx 的值为 0x00A97088，%edx 的值为 0x3003C088。

```
movl $0xC, (%ebx)
movl $0x9, (%edx)
movl (%ebx), %eax
xorl (%edx), %eax
```

【答】%ebx 与%edx 指向同一物理内存，因此答案为 0。

PART D. 下列 IA32 汇编代码执行结束以后，%eax 的值是多少？

```
movl 0xC0002A5C, %eax
```

重点关注加粗的内存。以此为启发，写出读出第一级页表中 VPN1=2 的条目的代码

```
movl 0xC0300008, %eax
```

【答】C0002A5C，VPN1=0x300，VPN2=0x2，因此 PDE 条目的物理地址为 0x00C188C00，读出第二级页表的起始地址为 0x00C188000（就是第一级页表!），PTE 条目起始地址为 0x00C188008，读出页地址为 0x00D32A000，物理地址为 0x00D32AA5C，因此%eax 的值为 0x9A83C067。为了指向第一级页表的第三个条目（VPN1=2），也就希望二级页表映射以后映射到的物理页就是第一级页表。因此前两次映射都应当映射到页表自己。发现如果 VPN1=0x300 的话，那么第一级映射就映射到自己了，于是如果 VPN2=0x300 的话，第二级映射也还是映射回自己。对应的 VPO=0x2*4=0x8，拼起来就是虚拟地址 0xC0300008。

【内存映射】

3．有下列程序：

```
int main() {
    pid_t pid;
    int child_status;
    long* f = mmap(NULL, 8,
           PROT_READ | PROT_WRITE,
           X | MAP_ANONYMOUS, -1, 0);
```

```
    *f = 0;
    if ((pid = fork()) > 0) {                // Parent
        waitpid(pid, &child_status, 0);
        *f = *f + 1;
        printf("Parent: %ld\n", *f);
    } else {                                 // Child
        *f = *f + 1;
        printf("Child: %ld\n", *f);
    }
    return 0;
}
```

当 X 处为 MAP_PRIVATE 时，标准输出上的两个整数是什么？如果是 MAP_SHARED 呢？

【答】1 1；1 2

4．有下列 C 程序。其中 sleep(3)是为了让 fork 以后子进程先运行。hello.txt 的初始内容为字符串 ABCDEFG，紧接着\0。LINUX 采用写时复制（Copy-on-Write）技术。

```
char* f;
int count = 0, parent = 0, child = 0, done = 0;
void handler1() {
    if (count >= 4) {
        done = 1;
        return;
    }
    f[count] = '0' + count;
    count ++;
    kill(parent, SIGUSR2);
}
void handler2() {
    [ Y ]
    write(STDOUT_FILENO, f, 7);
    write(STDOUT_FILENO, "\n", 1);
    kill(child, SIGUSR1);
}
int main() {
    signal(SIGUSR1, handler1); signal(SIGUSR2, handler2);
    int child_status;
    parent = getpid();
    int fd = open("hello.txt", O_RDWR);
    if ((child = fork()) > 0) {              // Parent
        f = mmap(NULL, 8, PROT_READ | PROT_WRITE,
               MAP_PRIVATE, fd, 0);
        sleep(3);
        kill(child, SIGUSR1);
        waitpid(child, &child_status, 0);
    } else {                                 // Child
```

```
        f = mmap(NULL, 8, PROT_READ | PROT_WRITE,
               MAP_SHARED, fd, 0);
        while (done == 0)
            ;
    }
    return 0;
}
```

(1) 若 Y 处为空。程序运行结束以后，标准输出上的内容是什么（四行）？hello.txt 中的内容是什么？

(2) 若 Y 处为 f[6] = 'X';。程序运行结束以后，标准输出上的内容是什么（四行）？hello.txt 中的内容是什么？

【答】(1) 0BCDEFG、01CDEFG、012DEFG、0123EFG，文件内容为 0123EFG (2)四行 0BCDEFX，文件内容为 0123EFG。注意只有写时才复制。

5．有如下 C 程序

```
int main() {
    char* hello;
    char* bye;
    int fd1 = open("hello.txt", O_RDWR);
    int fd2 = open("bye.txt", O_RDWR);
    hello = mmap(NULL, 16, PROT_READ,
        MAP_SHARED, fd1, 0);
    bye = mmap(NULL, 16, PROT_READ | PROT_WRITE,
        MAP_SHARED, fd2, 0);
    for (int i = 0; i < 8; i++)
        bye[i] = toupper(hello[i]);
    /************A**************/
    munmap(hello, 16);
    munmap(bye, 16);
    return 0;
}
```

代码运行到 A 处的时候，/proc/2333/maps 中的内容如下：

ADDRESS	PERM	PATH
00400000-00401000	(1)	/home/pw384/map/pstate
00600000-00601000	r--p	/home/pw384/map/pstate
00601000-00602000	rw-p	/home/pw384/map/pstate
7fb596fb5000-7fb59719c000	r-xp	/lib/x86_64-linux-gnu/libc-2.27.so
7fb59719c000-7fb59739c000	---p	/lib/x86_64-linux-gnu/libc-2.27.so
7fb59739c000-7fb5973a0000	r--p	/lib/x86_64-linux-gnu/libc-2.27.so
7fb5973a0000-7fb5973a2000	rw-p	/lib/x86_64-linux-gnu/libc-2.27.so
7fb5973a2000-7fb5973a6000	rw-p	
7fb5973a6000-7fb5973cd000	r-xp	/lib/x86_64-linux-gnu/(4)-2.27.so
7fb5975b1000-7fb5975b3000	rw-p	
7fb5975cb000-7fb5975cc000	(2)	/home/pw384/map/bye.txt

```
7fb5975cc000-7fb5975cd000  (3)     /home/pw384/map/hello.txt
7fb5975cd000-7fb5975ce000  r--p    /lib/x86_64-linux-gnu/(4)-2.27.so
7fb5975ce000-7fb5975cf000  rw-p    /lib/x86_64-linux-gnu/(4)-2.27.so
7fb5975cf000-7fb5975d0000  rw-p
7ffe671ef000-7ffe67210000  rw-p    [(5)]
7ffe673cc000-7ffe673cf000  r--p    [vvar]
7ffe673cf000-7ffe673d1000  r-xp    [vdso]
ffffffffff600000-ffffffffff601000 r-xp [vsyscall]
```

PERM 有四位。前三位是 r=readable，w=writeable，x=executable，如果是-表示没有这一权限。第四位是 s=shared，p=private，表示映射是共享的还是私有的。填写空格的内容：

1: r-xp

2: rw-s

3: r--s

4: ld

5: stack

7fb5973a0000 对应的页在页表中被标为了只读。对该页进行写操作会发生 SIGSEGV 吗？

【答】不会，在内核的映射表中标出它是可写的，因此这是 COW 还未发生的情况。