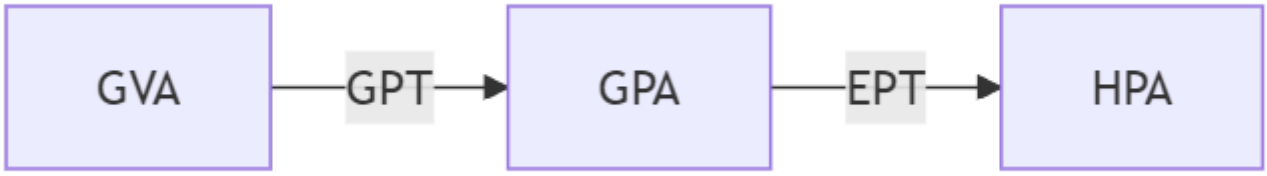
实验二: 内存虚拟化

[522031910557] [冯海桐]

1 调研部分

1. 在现代操作系统中，虚拟内存和物理内存被分为 4KB 页，并且将虚拟页号对应的物理页号记录在映射表中。页表又被组成多级页表，用以减少内存占用。

2. EPT 地址翻译过程的示意图如下：



3. 首先调用 `generate_memory_topology` 函数生成其 physmr 根对应的 FlatView，再调用函数 `address_space_set_flatview`→`address_space_update_topology_pass`→`kvm_region_add` 通知KVM模块：线性视图已经更改，需要重新向KVM使用 `ioctl` 函数注册内存，然后调用 `ioctl` 函数进入内核态的KVM模块中，调用 `KVM_SET_TSS_ADDR` 和 `KVM_SET_IDENTITY_MAP_ADDR` 设置相关地址，以此建立 EPT。

2 实验目的

1. 理解内存虚拟化的基本概念；
2. 理解内存虚拟化的基本原理；
3. 打印地址翻译过程，并加以分析。

3 实验步骤

1. 启动并登陆到 L1 虚拟机中。
2. 进入 L1 虚拟机的 mem_lab 目录下：`$ cd ~/labs/mem_lab`。
3. 启动并登陆到 L2 虚拟机中。
4. 通过 `$ make` 编译目标程序，通过 `sudo ./run.sh` 运行后，保存 gpt-dump.txt，此为客户机页表的翻译过程。
5. 回到 L1 虚拟机中，通过 `sudo dmesg` 打印 GPA 到 HPA 的 EPT 翻译过程。
6. 根据记录的翻译过程进行分析，描述 GVA 到 HPA 的翻译过程。

4 实验分析

4.1 代码理解

进行宏定义，用于将二进制数据转换成字符串，以便打印输出。

```
#define BYTE_TO_BINARY(byte) \
  ((byte) & 0x80 ? '1' : '0'), ((byte) & 0x40 ? '1' : '0'),                   \
  ((byte) & 0x20 ? '1' : '0'), ((byte) & 0x10 ? '1' : '0'),                   \
  ((byte) & 0x08 ? '1' : '0'), ((byte) & 0x04 ? '1' : '0'),                   \
  ((byte) & 0x02 ? '1' : '0'), ((byte) & 0x01 ? '1' : '0')

// 其他宏定义省略...
```

接下来定义了如 `pr_pte`、`print_ptr_vaddr` 的打印函数。

接着定义了页表遍历函数，用于遍历 PGD、PUD、PMD、PTE，并打印相关信息。

```
void dump_pgd(pgd_t *pgtable, int level) {
    unsigned long i;
    pgd_t pgd;
    pr_sep();

    for (i = 0; i < PTRS_PER_PGD; i++) {
        pgd = pgtable[i];

        if (pgd_val(pgd)) {
            if (i == pgd_idx) {
                if (pgd_large(pgd)) {
                    pr_info("Large pgd detected! return"); break;
                }
                if (pgd_present(pgd)) {
                    pr_pte(__pa(pgtable), pgd_val(pgd), i, level);
                    dump_pud((pud_t *) pgd_page_vaddr(pgd), level + 1);
                }
            }
        }
    }
}

// 其他页表遍历函数省略...
```

`init_module` 函数是模块的入口点，分配内存并初始化一个指针，然后打印虚拟地址和物理地址，并调用页表遍历函数。

```
int init_module(void) {
    volatile unsigned long *ptr;
    int i;

    ptr = kmalloc(sizeof(int), GFP_KERNEL);
    for (i = 0; i < 1; ++i)
      ptr[i] = i*i;
    *ptr = 1772333;
    printk("Value at GVA: %lu", ++*ptr);
```

```
    print_ptr_vaddr(ptr);
    dump_pgd(current->mm->pgd, 1);
    print_pa_check(vaddr);

    kvm_hypercall1(22, paddr);
    for (i = 0; i < 1; ++i)
      ptr[i] = ptr[i] - 1;
    kfree((const void *) ptr);

    return 0;
}
```

这段代码在客户机上运行，实现了将 GVA 翻译成 GPA 的过程，并将翻译后的地址通过 `kvm_hypercall1` 函数，执行 KVM 超级调用，传递给主机。主机进一步把 GPA 翻译成 HPA，把相应的数值写入到目的地址。

4.2 实验结果

运行的结果文本如附录2、3所示。

4.2.1 GPT

GVA 的地址结构为 9+9+9+9+12，分别对应 PGD、PUD、PMD 的 index 和 GPA 的 Offset。查找 GPA 时，首先从控制寄存器 CR3 中读取 GPT 的物理基地址。由于页表项（PTE）的大小为 8B (文件中的 64 应当为 8B)，所以对应的 PGD 的地址应该是 CR3 中基地址加上 GVA[47 : 19] * 8B 中所保存的内容。同理，可以得到 PUD、PMD 和 PTE 的地址，进而最终得到 GPA。以上过程由客户机完成。

4.2.2 EPT

GPA 由客户机传到主机，主机将其进一步翻译成 HPA。GPA 的地址结构也是 9+9+9+9+12，其前面的翻译过程与上文一样。在 PMD 层级，由于检测到了 2M 大页，因此从 PMD 到 HPA 的过程需要掩码。PMD 的结构为 12+31+9+12，分别对应保留或忽略位、页框号、忽略位和 Offset，掩码提取出 PMD 的页框号，并将其与 HPA 的 Offset 结合，最终得到 HPA。

最终我们可以看到，HPA 中的值是 1772334，和最初写入到 GVA 的时候一致，完成了 GVA 到 HPA 的翻译。

5 遇到的问题及解决方案

在 markdown 中可以显示的图片，导出为 pdf 后却无法显示，通过截图并插入后解决。

附录 1

实验代码

```
#include <linux/cpu.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/mm.h>
#include <linux/module.h>
```

```
#include <linux/sched.h>
#include <linux/slab.h>
#include <linux/sort.h>
#include <linux/string.h>

#include <asm/pgtable.h>
#include <asm/uaccess.h>
#include <asm/kvm_para.h>

MODULE_LICENSE("GPL");

/* convert to 0, 0, 1, 0, 1, 1, 0, ... */

// convert unsigned long to vaddr
#define BYTE_TO_BINARY(byte) \
  ((byte) & 0x80 ? '1' : '0'), ((byte) & 0x40 ? '1' : '0'),                    \
  ((byte) & 0x20 ? '1' : '0'), ((byte) & 0x10 ? '1' : '0'),                    \
  ((byte) & 0x08 ? '1' : '0'), ((byte) & 0x04 ? '1' : '0'),                    \
  ((byte) & 0x02 ? '1' : '0'), ((byte) & 0x01 ? '1' : '0')

#define TBYTE_TO_BINARY(tbyte)  \
  ((tbyte) & 0x04 ? '1' : '0'),    \
  ((tbyte) & 0x02 ? '1' : '0'), ((tbyte) & 0x01 ? '1' : '0')

#define UL_TO_PTE_OFFSET(ulong) \
  TBYTE_TO_BINARY((ulong) >> 9), TBYTE_TO_BINARY((ulong) >> 6), \
  TBYTE_TO_BINARY((ulong) >> 3), TBYTE_TO_BINARY((ulong))

#define UL_TO_PTE_INDEX(ulong) \
  TBYTE_TO_BINARY((ulong) >> 6), TBYTE_TO_BINARY((ulong) >> 3),
TBYTE_TO_BINARY((ulong))

#define UL_TO_VADDR(ulong) \
  UL_TO_PTE_INDEX((ulong) >> 39), UL_TO_PTE_INDEX((ulong) >> 30), \
  UL_TO_PTE_INDEX((ulong) >> 21), UL_TO_PTE_INDEX((ulong) >> 12), \
  UL_TO_PTE_OFFSET((ulong))


// convert unsigned long to pte
#define UL_TO_PTE_PHYADDR(ulong) \
  BYTE_TO_BINARY((ulong) >> 32),    \
  BYTE_TO_BINARY((ulong) >> 24), BYTE_TO_BINARY((ulong) >> 16), \
  BYTE_TO_BINARY((ulong) >> 8), BYTE_TO_BINARY((ulong) >> 0)

#define UL_TO_PTE_IR(ulong) \
  UL_TO_PTE_OFFSET(ulong)

#define UL_TO_PTE(ulong) \
  UL_TO_PTE_IR((ulong) >> 52), UL_TO_PTE_PHYADDR((ulong) >> PAGE_SHIFT),
UL_TO_PTE_OFFSET(ulong)

#define UL_TO_PADDR(ulong) \
  UL_TO_PTE_PHYADDR((ulong) >> PAGE_SHIFT), UL_TO_PTE_OFFSET(ulong)
```

```
/* printk pattern strings */

// convert unsigned long to vaddr
#define TBYTE_TO_BINARY_PATTERN   "%c%c%c"
#define BYTE_TO_BINARY_PATTERN    "%c%c%c%c%c%c%c%c"

#define PTE_INDEX_PATTERN \
  TBYTE_TO_BINARY_PATTERN TBYTE_TO_BINARY_PATTERN TBYTE_TO_BINARY_PATTERN " "

#define VADDR_OFFSET_PATTERN \
  TBYTE_TO_BINARY_PATTERN TBYTE_TO_BINARY_PATTERN \
  TBYTE_TO_BINARY_PATTERN TBYTE_TO_BINARY_PATTERN

#define VADDR_PATTERN \
  PTE_INDEX_PATTERN PTE_INDEX_PATTERN \
  PTE_INDEX_PATTERN PTE_INDEX_PATTERN \
  VADDR_OFFSET_PATTERN


// convert unsigned long to pte
// 40 bits
#define PTE_PHYADDR_PATTREN \
  BYTE_TO_BINARY_PATTERN BYTE_TO_BINARY_PATTERN \
  BYTE_TO_BINARY_PATTERN BYTE_TO_BINARY_PATTERN \
  BYTE_TO_BINARY_PATTERN " "

// 12 bits
#define PTE_IR_PATTERN \
  VADDR_OFFSET_PATTERN " "

// 12 + 40 + 12 bits
#define PTE_PATTERN \
  PTE_IR_PATTERN PTE_PHYADDR_PATTREN VADDR_OFFSET_PATTERN

#define PADDR_PATTERN \
  PTE_PHYADDR_PATTREN VADDR_OFFSET_PATTERN

static inline void pr_sep(void) {
    // pr_err("
.....................................................................\n");
    pr_err("\n");
}

/* static vals */
unsigned long vaddr, paddr, pgd_idx, pud_idx, pmd_idx, pte_idx;
const char *PREFIXES[] = {"PGD", "PUD", "PMD", "PTE"};

/* static inline functions */
static inline void pr_pte(unsigned long address, unsigned long pte,
                          unsigned long i, int level) {
    if (level == 1)
        pr_cont(" NEXT_LVL_GPA(CR3)  =  ");
    else
        pr_cont(" NEXT_LVL_GPA(%s)  =  ", PREFIXES[level - 2]);
```

```
    pr_cont(PTE_PHYADDR_PATTREN, UL_TO_PTE_PHYADDR(address >> PAGE_SHIFT));
    pr_cont(" +  8 * %-3lu\n", i);
    pr_sep();

    pr_cont(" %-3lu: %s " PTE_PATTERN"\n", i, PREFIXES[level - 1],
UL_TO_PTE(pte));
}

static inline void print_ptr_vaddr(volatile unsigned long *ptr) {
    unsigned long mask = ((1 << 9) - 1);

    vaddr = (unsigned long) ptr;
    pgd_idx = (vaddr >> 39) & mask;
    pud_idx = (vaddr >> 30) & mask;
    pmd_idx = (vaddr >> 21) & mask;
    pte_idx = (vaddr >> 12) & mask;
    pr_info(" GPT PGD index: %lu", pgd_idx);
    pr_info(" GPT PUD index: %lu", pud_idx);
    pr_info(" GPT PMD index: %lu", pmd_idx);
    pr_info(" GPT PTE index: %lu", pte_idx);

    pr_info("          %lu       %lu       %lu       %lu", pgd_idx, pud_idx,
pmd_idx, pte_idx);
    pr_info(" GVA [PGD IDX] [PUD IDX] [PMD IDX] [PTE IDX] [  Offset  ]");
    pr_info(" GVA "VADDR_PATTERN"\n", UL_TO_VADDR(vaddr));
}

static inline void print_pa_check(unsigned long vaddr) {
    paddr = __pa(vaddr);
    pr_info(" GPA            =      " PADDR_PATTERN "\n", UL_TO_PADDR(paddr));
}

/* page table walker functions */
void dump_pgd(pgd_t *pgtable, int level);

void dump_pud(pud_t *pgtable, int level);

void dump_pmd(pmd_t *pgtable, int level);

void dump_pte(pte_t *pgtable, int level);

int init_module(void) {
    volatile unsigned long *ptr;
    int i;

    ptr = kmalloc(sizeof(int), GFP_KERNEL);
    for (i = 0; i < 1; ++i)
      ptr[i] = i*i;
    *ptr = 1772333;
    printk("Value at GVA: %lu", ++*ptr);

    print_ptr_vaddr(ptr);
    dump_pgd(current->mm->pgd, 1);
    print_pa_check(vaddr);
```

```
    kvm_hypercall1(22, paddr);
    for (i = 0; i < 1; ++i)
      ptr[i] = ptr[i] - 1;
    kfree((const void *) ptr);

    return 0;
}

void cleanup_module(void) {}

void dump_pgd(pgd_t *pgtable, int level) {
    unsigned long i;
    pgd_t pgd;
    pr_sep();


    for (i = 0; i < PTRS_PER_PGD; i++) {
        pgd = pgtable[i];

        if (pgd_val(pgd)) {
            if (i == pgd_idx) {
                if (pgd_large(pgd)) {
                    pr_info("Large pgd detected! return"); break;
                }
                if (pgd_present(pgd)) {

                    pr_pte(__pa(pgtable), pgd_val(pgd), i, level);

                    dump_pud((pud_t *) pgd_page_vaddr(pgd), level + 1);
                }
            }
        }
    }
}

void dump_pud(pud_t *pgtable, int level) {
    unsigned long i;
    pud_t pud;

    for (i = 0; i < PTRS_PER_PUD; i++) {
        pud = pgtable[i];

        if (pud_val(pud)) {
            if (i == pud_idx) {
                if (pud_large(pud)) {
                    pr_info("Large pud detected! return"); break;
                }
                if (pud_present(pud) && !pud_large(pud)) {

                    pr_pte(__pa(pgtable), pud_val(pud), i, level);

                    dump_pmd((pmd_t *) pud_page_vaddr(pud), level + 1);
                }
```

```
            }
        }
    }
}

void dump_pmd(pmd_t *pgtable, int level) {
    unsigned long i;
    pmd_t pmd;

    for (i = 0; i < PTRS_PER_PMD; i++) {
        pmd = pgtable[i];

        if (pmd_val(pmd)) {
            if (i == pmd_idx) {

                if (pmd_large(pmd)) {
                    pr_info("Large pmd detected! return"); break;
                }
                if (pmd_present(pmd) && !pmd_large(pmd)) {
                    pr_pte(__pa(pgtable), pmd_val(pmd), i, level);

                    dump_pte((pte_t *) pmd_page_vaddr(pmd), level + 1);
                }
            }
        }
    }
}

void dump_pte(pte_t *pgtable, int level) {
    unsigned long i;
    pte_t pte;

    for (i = 0; i < PTRS_PER_PTE; i++) {
        pte = pgtable[i];

        if (pte_val(pte)) {
            if (pte_present(pte)) {
                if (i == pte_idx)
                    pr_pte(__pa(pgtable), pte_val(pte), i, level);
            }
        }
    }
}
```

附录 2

GPT

```
Value at GVA: 1772334
 GPT PGD index: 320
 GPT PUD index: 13
```

```
GPT PMD index: 453
GPT PTE index: 100
         320       13       453       100
GVA [PGD IDX] [PUD IDX] [PMD IDX] [PTE IDX] [  Offset  ]
GVA 101000000 000001101 111000101 001100100 010110001000

NEXT_LVL_GPA(CR3)  =  000000000000000000001010011100000101110  +  64 * 320

320: PGD 000000000000 000000000000000000001110000010110011111 000001100111
NEXT_LVL_GPA(PGD)  =  000000000000000000001110000010110011111  +  64 * 13

13 : PUD 000000000000 000000000000000000001110000010110100011 000001100111
NEXT_LVL_GPA(PUD)  =  000000000000000000001110000010110100011  +  64 * 453

453: PMD 000000000000 000000000000000000001101100110000011110 000001100011
NEXT_LVL_GPA(PMD)  =  000000000000000000001101100110000011110  +  64 * 100

100: PTE 100000000000 000000000000000000001111000101001100100 000001100011
GPA            =      000000000000000000001111000101001100100 010110001000
```

附录 3

EPT

```
[  322.151149]  EPT PGD index: 0
[  322.151155]  EPT PUD index: 1
[  322.151155]  EPT PMD index: 453
[  322.151156]  EPT PTE index: 100
[  322.151157]           0       1       453       100
[  322.151159]  GPA [PGD IDX] [PUD IDX] [PMD IDX] [PTE IDX] [  Offset  ]
[  322.151160]  GPA 000000000 000000001 111000101 001100100 010110001000
[  322.151164] This is EPT

[  322.151166]  NEXT_LVL_HPA(EPTP) =  000000000000000000000000110000001111111  +
64 * 0

[  322.151190]  0  : PGD 000000000000 000000000000000000000000110000001111110
100100000111
[  322.151193]  NEXT_LVL_HPA(PGD)  =  000000000000000000000000110000001111110  +
64 * 1

[  322.151202]  1  : PUD 000000000000 000000000000000000000000110000001111101
100100000111
[  322.151205]  NEXT_LVL_HPA(PUD)  =  000000000000000000000000110000001111101  +
64 * 453

[  322.151212]  453: PMD 000001100000 000000000000000000000000001101011000000000
101111110111
[  322.151215] Huge Page (2M) at level 2 detected!
[  322.151217] [e.g]   [Rsvd./Ign.] [  Huge Page Number, 31 Bits  ][Ignored] [
Flags  ]
```

```
[  322.151218]  PMD    000001100000 0000000000000000000000001101011000000000
101111110111 (PMD Entry t)
[  322.151226]  Mask1  000000000000 1111111111111111111111111111111000000000
000000000000 (get HPA of )
[  322.151228]  HFN    000000000000 0000000000000000000000001101011000000000
000000000000 (HFN of Huge)
[  322.151230] ----------------------------------------------------------------
--------------------
[  322.151232]  GPA    000000000000 0000000000000000000001111000101001100100
010110001000 (GPA from Gu)
[  322.151260]  Mask2  000000000000 0000000000000000000000000000111111111
111111111111 (get offset )
[  322.151262]  Offset 000000000000 0000000000000000000000000000001100100
010110001000 (offset in H)
[  322.151264] ----------------------------------------------------------------
--------------------
[  322.151265] HPA   000000000000 0000000000000000000000001101011001100100
010110001000
[  322.151292] Value at HPA: 1772334
[  416.156520]  EPT PGD index: 0
[  416.156525]  EPT PUD index: 1
[  416.156526]  EPT PMD index: 453
[  416.156527]  EPT PTE index: 100
[  416.156527]           0       1       453       100
[  416.156530]  GPA [PGD IDX] [PUD IDX] [PMD IDX] [PTE IDX] [  Offset  ]
[  416.156531]  GPA 000000000 000000001 111000101 001100100 010110001000
[  416.156536] This is EPT

[  416.156537]  NEXT_LVL_HPA(EPTP) =  0000000000000000000000000110000001111111  +
64 * 0

[  416.156559]  0  : PGD 000000000000 0000000000000000000000000110000001111110
100100000111
[  416.156563]  NEXT_LVL_HPA(PGD)  =  0000000000000000000000000110000001111110  +
64 * 1

[  416.156572]  1  : PUD 000000000000 0000000000000000000000000110000001111101
100100000111
[  416.156575]  NEXT_LVL_HPA(PUD)  =  0000000000000000000000000110000001111101  +
64 * 453

[  416.156583]  453: PMD 000001100000 0000000000000000000000001101011000000000
101111110111
[  416.156586] Huge Page (2M) at level 2 detected!
[  416.156588] [e.g]   [Rsvd./Ign.] [  Huge Page Number, 31 Bits  ][Ignored] [
Flags  ]
[  416.156589]  PMD    000001100000 0000000000000000000000001101011000000000
101111110111 (PMD Entry t)
[  416.156592]  Mask1  000000000000 1111111111111111111111111111111000000000
000000000000 (get HPA of )
[  416.156595]  HFN    000000000000 0000000000000000000000001101011000000000
000000000000 (HFN of Huge)
[  416.156597] ----------------------------------------------------------------
--------------------
```

```
[  416.156598]  GPA    000000000000 000000000000000000000111100010100110010
010110001000 (GPA from Gu)
[  416.156629]  Mask2  000000000000 000000000000000000000000000111111111
111111111111 (get offset )
[  416.156632]  Offset 000000000000 000000000000000000000000000001100100
010110001000 (offset in H)
[  416.156634] ------------------------------------------------------------------
--------------------
[  416.156635] HPA   000000000000 000000000000000000000011010110011001000
010110001000
[  416.156672] Value at HPA: 1772334
[  615.341107] loop6: detected capacity change from 0 to 130448
```