作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师
https://github.com/drink-cat/Book_Program-Principles

内存地址与页表

基础知识
整数分为有符号整数、无符号整数。
有符号整数最高位表示符号位，表示整数值的位数会少一位，可以表示正数、负数、0。比如，32 位有符号整数，代码为 `signed int`，简写为 `i32`。
无符号整数最高位不表示符号位，全部位数都表示整数值，可以表示正数、0。比如，32 位无符号整数，代码为 `unsigned int`，简写为 `u32`。
i32 与 u32，i 前缀表示 signed int，u 前缀表示 unsigned int，数字 32 表示 32 位。

本节讲解虚拟内存地址。内存地址，使用无符号整数表示，比如 64 位 CPU 使用 u64 表示内存地址。
u64 表示的整数范围非常大，实际上，内存地址不需要那么大的整数上限。所以使用部分低位表示内存地址，剩余的高位表示其他功能，比如读写权限。
64 位 CPU 使用 48 位表示内存地址。48 位无符号整数最大值为 2 的 48 次方，表示 256TB 内存空间。
从系统安全、内存共享等角度考虑，又把内存空间分为用户空间、内核空间。用户空间使用低地址的 128TB，内核空间使用高地址的 128TB。

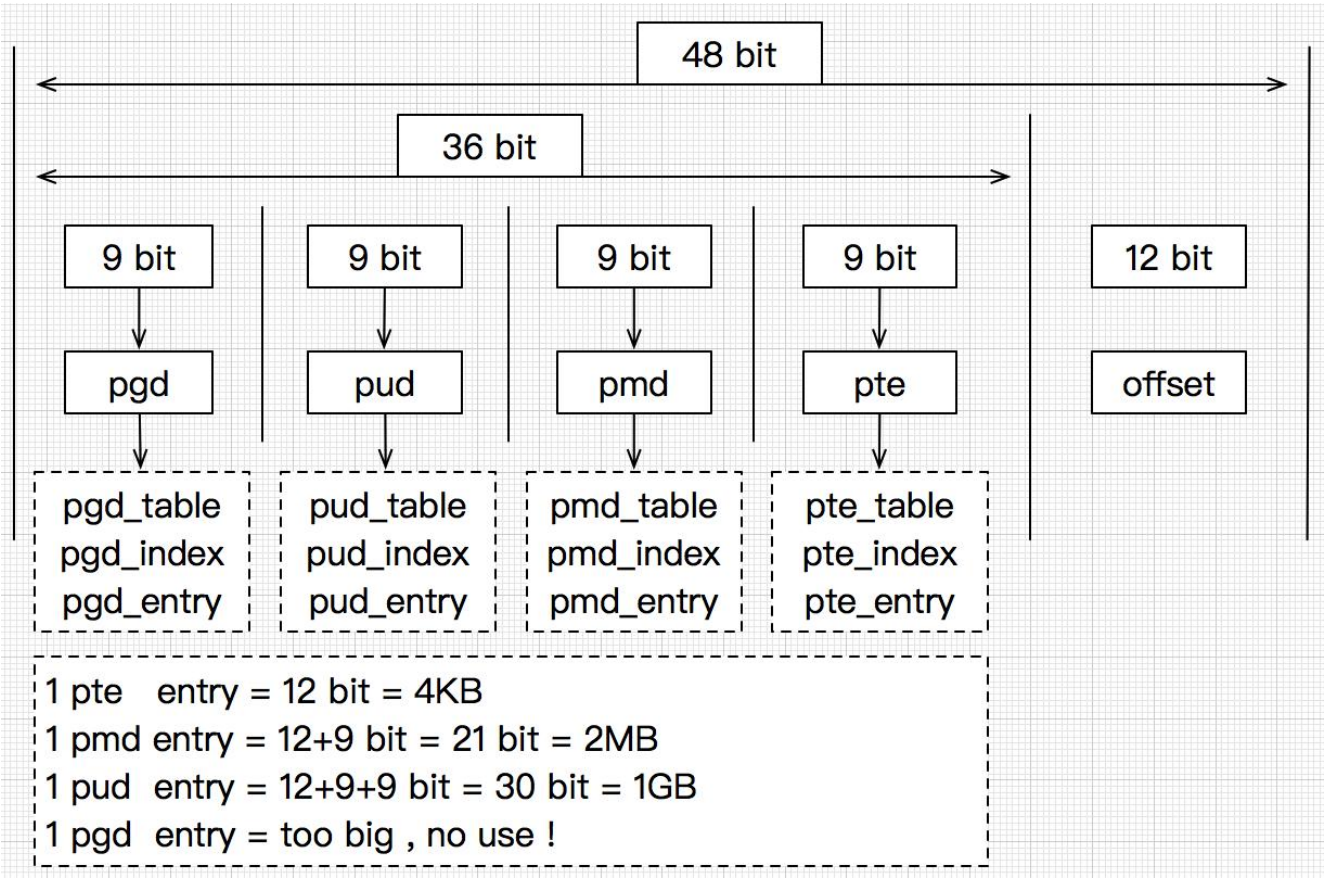
内存分为虚拟内存、物理内存。物理内存受限于内存条的大小，比如常见的 PC 机使用 8G 或 16G 物理内存。同时，虚拟内存很大，CPU 和进程操作虚拟内存，不直接操作物理内存。
从虚拟内存到物理内存需要有映射。比如 CPU 读取某个变量的值，首先找到变量所在的虚拟内存地址，然后使用映射用虚拟内存地址找到物理内存地址，最后读取物理内存地址对应的变量值。
映射必须满足结构紧凑、占用内存少、读写效率高、易于缓存等要求。

页表实现虚拟内存到物理内存的映射，是虚拟内存的重要部分。
页表的技术实现为前缀树+数组，参考下文的页表示意图。
前缀树用于把内存地址分级，构成多级页表，上一级节点指向多个下一级节点，查询页表时依次遍历每一级节点。
数组用于表示每个层级节点，一个数组表示一批连续的虚拟内存地址，优点为节约内存、读写效率高。

linux 常用 4 级页表。48 位虚拟内存地址切分为 pgd、pud、pmd、pte、offset，其中 pgd、pud、pmd、pte 组成页表。
pgd 表示 Page Global Directory，即全局目录项。位于最上层。
pud 表示 Page Upper Directory，即上级目录项。位于 pgd 与 pmd 之间。
pmd 表示 Page Middle Directory，即中间目录项。位于 pud 与 pte 之间。
pte 表示 Page Table Entry，即页表项。位于最下层。
offset 表示页内偏移，即在 pte 节点中的地址偏移量。
可以用下图方便记忆：

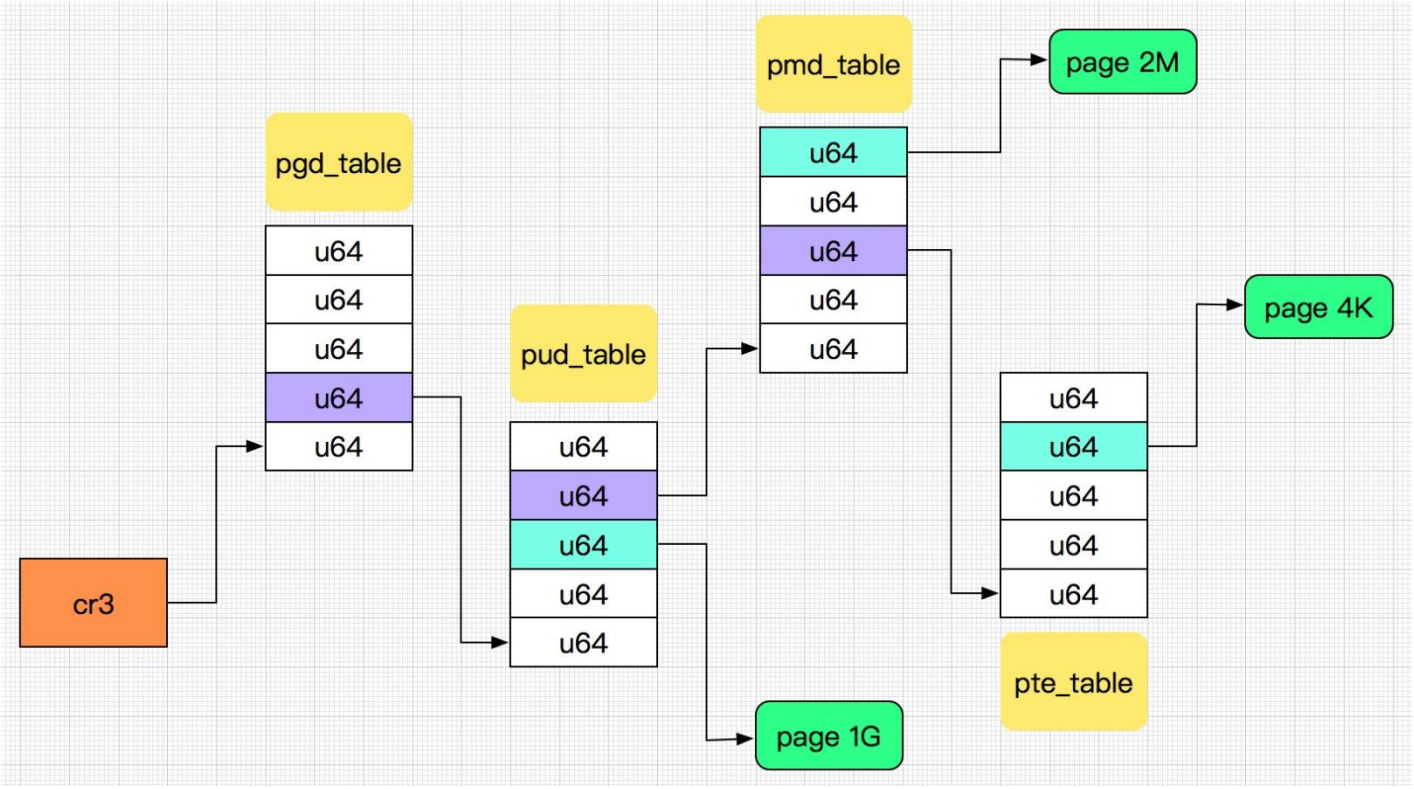
```
pgd  9 位  global   全局
pud  9 位  upper    上层
pmd  9 位  middle   中层
pte  9 位  entry    元素
offset 12 位  页内偏移。
pgd、pud、pmd、pte 组成 4 级页表。
```

4 级页表的示意图：



页表的本质是层级和数组。
4 级页表分为 4 个层级，每个层级都用数组表示。
pgd 数组的每一项，指向一个 pud 数组。
pud 数组的每一项，指向一个 pmd 数组，或者指向一个 1G 大页。
pmd 数组的每一项，指向一个 pte 数组，或者指向一个 2M 大页。
pte 数组的每一项，指向一个 4K 小页。

页表层级的示意图：



问题：页面大小 4K、2M、1G，怎么计算出来的？
12 位，表示 2 的 12 次方，4K 大小。
(12+9)位，表示 2 的 21 次方，2M 大小。
(12+9+9)位，表示 2 的 30 次方，1G 大小。

问题：为什么区分小页、大页？
小页表示大小为 4K 的页，由 pte 数组管理。小页最常用，一方面程序最常分配小块内存，一方面读写文件用作页缓存。
大页表示大于 4K 的页，包括 2M 大页、1G 大页，由 pmd 数组、pud 数组管理。大页用于需要大块内存的场景。
假设需要分配 2M 内存。
如果使用小页，则需要一个完整的 pte 数组来表示。
如果使用大页，则只需要 pmd 数组的一个元素来表示，不需要 pte 数组。
可见，大页的作用为，节约 pte 数组所占用的内存，也简化了分配内存、是否内存的操作。

模拟页表功能

编写代码： page_table.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 各个部分占用的位数。一共 48 位。
int bit_offset = 12;
int bit_pte = 9;
int bit_pmd = 9;
```

```
int bit_pud = 9;
int bit_pgd = 9;

// 移位的位数。用于计算每一部分的值。
int shift_offset = -1;
int shift_pte = -1;
int shift_pmd = -1;
int shift_pud = -1;
int shift_pgd = -1;

// 掩码。用于计算每一部分的值。
#define _build_mask(off) (1 << off) - 1
int mask_offset = -1;
int mask_pte = -1;
int mask_pmd = -1;
int mask_pud = -1;
int mask_pgd = -1;

// 初始化。
void init_all()
{
    shift_offset = 0;
    shift_pte = shift_offset + bit_offset;
    shift_pmd = shift_pte + bit_pte;
    shift_pud = shift_pmd + bit_pmd;
    shift_pgd = shift_pud + bit_pud;
    mask_offset = _build_mask(bit_offset);
    mask_pte = _build_mask(bit_pte);
    mask_pmd = _build_mask(bit_pmd);
    mask_pud = _build_mask(bit_pud);
    mask_pgd = _build_mask(bit_pgd);
}

// 从虚拟地址取出一个部分。
int fetch_part(unsigned long long virtual_addr, int shift_xx, int mask_xx)
{
    // 移位。让该部分靠右。
    unsigned long long tmp = virtual_addr >> shift_xx;
    // 位与。只保留该部分。
    unsigned long long tmp2 = tmp & mask_xx;
    return (int)tmp2;
}

// 从虚拟地址截取各个部分。
#define fetch_offset(va) fetch_part(va, shift_offset, mask_offset)
#define fetch_pte(va) fetch_part(va, shift_pte, mask_pte)
#define fetch_pmd(va) fetch_part(va, shift_pmd, mask_pmd)
#define fetch_pud(va) fetch_part(va, shift_pud, mask_pud)
#define fetch_pgd(va) fetch_part(va, shift_pgd, mask_pgd)
```

```
// 把整数转成 2 进制。指定多少位。高位在左侧。
void cvt_bin(unsigned long long num, char *buf, int bit_count)
{
    // 高位在左侧。依次取一个 bit 。
    char *buf2 = buf;
    for (int count = bit_count - 1; count >= 0; --count)
    {
        // 右移。
        unsigned long long tmp = num >> count;
        // 截取一个 bit。
        int bit = (int)(tmp & 1);
        // 保留一个字符
        char ch = (bit == 0) ? '0' : '1';
        *buf2 = ch;
        ++buf2;
    }
    // 字符串的末尾。
    *buf2 = '\0';
}

int main()
{
    init_all();

    printf("\n 查看页表的各个部分：\n");
    printf("%10s  %5s  %5s  %6s \n", "PART", "bits", "shift", "mask");
    printf("%10s  %5d  %5d  %#6x \n", "pgd", bit_pgd, shift_pgd, mask_pgd);
    printf("%10s  %5d  %5d  %#6x \n", "pud", bit_pud, shift_pud, mask_pud);
    printf("%10s  %5d  %5d  %#6x \n", "pmd", bit_pmd, shift_pmd, mask_pmd);
    printf("%10s  %5d  %5d  %#6x \n", "pte", bit_pte, shift_pte, mask_pte);
    printf("%10s  %5d  %5d  %#6x \n", "offset", bit_offset, shift_offset, mask_offset);

    // 一个虚拟地址。48 位，使用 6 个字节。
    unsigned long long virtual_addr = 0x12F3F4F5F6F7;
    // 拆分出各个部分
    int part_offset = fetch_offset(virtual_addr);
    int part_pte = fetch_pte(virtual_addr);
    int part_pmd = fetch_pmd(virtual_addr);
    int part_pud = fetch_pud(virtual_addr);
    int part_pgd = fetch_pgd(virtual_addr);
    // 查看各个部分的二进制
    char buf_va[50], buf_offset[13], buf_pte[10], buf_pmd[10], buf_pud[10], buf_pgd[10];
    cvt_bin(virtual_addr, buf_va, 48);
    cvt_bin(part_offset, buf_offset, bit_offset);
    cvt_bin(part_pte, buf_pte, bit_pte);
    cvt_bin(part_pmd, buf_pmd, bit_pmd);
    cvt_bin(part_pud, buf_pud, bit_pud);
    cvt_bin(part_pgd, buf_pgd, bit_pgd);
```

```
    // 为了便于查看，把二进制的位对齐。
    printf("\n 查看虚拟地址的各个部分：\n");
    printf("virtual_addr  %20llu  %50s \n", virtual_addr, buf_va);
    printf("part_pgd      %20d  %11s \n", part_pgd, buf_pgd);
    printf("part_pud      %20d  %20s \n", part_pud, buf_pud);
    printf("part_pmd      %20d  %29s \n", part_pmd, buf_pmd);
    printf("part_pte      %20d  %38s \n", part_pte, buf_pte);
    printf("part_offset   %20d  %50s \n", part_offset, buf_offset);

    printf("\n\n");
    return 0;
}
```

编译代码：

```
# 编译为可执行程序
gcc page_table.c -o page_table -std=gnu99
```

运行代码：

```
[root@192 mem]# ./page_table

查看页表的各个部分：
      PART   bits  shift    mask
       pgd      9     39   0x1ff
       pud      9     30   0x1ff
       pmd      9     21   0x1ff
       pte      9     12   0x1ff
    offset     12      0   0xfff

查看虚拟地址的各个部分：
virtual_addr        20838996113143    000100101111001111110100111101011111011011110111
part_pgd                        37    000100101
part_pud                       463            111001111
part_pmd                       423                    110100111
part_pte                       351                            101011111
part_offset                   1783                                    011011110111
```

分析结果：

48 位地址，占用 6 个字节。

各个部分，分别占用多个位，使用位拆分。打印二进制分析位的对应关系。

高位在左侧。从左往右依次拆分出各个部分。

虚拟地址，48 个 bit，000100101111001111110100111101011111011011110111。

pgd 部分，9 个 bit，000100101。

pud 部分，9 个 bit，111001111。

pmd 部分，9 个 bit，110100111。

pte 部分，9 个 bit，101011111。

offset 部分，12 个 bit，011011110111。