Lab2 实验报告

19373135 田旗舰

一、实验思考题

Thinking 2.1

请思考 cache 用虚拟地址来查询的可能性，并且给出这种方式对访存带来的好处和坏处。另外，你能否能根据前一个问题的解答来得出用物理地址来查询的优势?

如果能够根据虚拟地址找到正确的物理地址，使用虚拟地址来查询是可能的。优点是查询前无需经过页表进行地址变换，缺点是一旦缺失，更新 cache 时也需要访问页表，时间代价更大，同时，不同程序的虚拟地址可能相同，正确性和安全性不易保证。

采用物理地址可以保证数据的正确性和安全性，也方便数据共享。

Thinking 2.2

在我们的实验中，有许多对虚拟地址或者物理地址操作的宏函数(详见 include/mmu.h),那么我们在调用这些宏的时候需要弄清楚需要操作的地址是物理地址还是虚拟地址，阅读下面的代码，指出 x 是一个物理地址还是虚拟地址。

```
int x;

char *value = return_a_pointer();

*value = 10;

x = (int) value;
```

x 是虚拟地址，因为 value 直接指向了具体的值。

Thinking 2.3

我们在 include/queue.h 中定义了一系列的宏函数来简化对链表的操作。实际上，我们在 include/queue.h 文件中定义的链表和 glibc 相关源码较为相似，这一链表设计也应用于 Linux 系统中 (sys/queue.h 文件)。请阅读这些宏函数的代码，说说它们的原理和巧妙之处。

些宏函数用于链表的创建和操作，包括表头定义、表头初始化、链表节点的结构体定义、清空表头、返回表头、链表遍历、链表初始化、节点后插入、节点前插入、表头前插入、链表尾插入、返回下一节点、删除节点。

链表头定义为结构体，内部为头指针，其后的节点结构体内部为下一项的地址和前一项中存放下一项的指针的地址。

巧妙之处是利用了宏定义，实现了链表名称、类型、内容的自定义，并实现了良好的封装，此外，不同与一般的链表，除了数据域和指向下一节点的指针之外，还存放了前一项中存放下一项的指针的地址，使得插入、删除操作更加方便。

Thinking 2.4

我们注意到我们把宏函数的函数体写成了 do { /* ... */ } while(0)的形式，而不是仅仅写成形如 { /* ... */ } 的语句块，这样的写法好处是什么？

封装更好，使其相当于一条语句而不是多条，避免使用中的歧义和错误。

Thinking 2.5

注意，我们定义的 Page 结构体只是一个信息的载体，它只代表了相应物理内存页的信息，它本身并不是物理内存页。 那我们的物理内存页究竟在哪呢？ Page 结构体又是通过怎样的方式找到它代表的物理内存页的地址呢？ 请你阅读 include/pmap.h 与 mm/pmap.c 中相关代码，并思考一下。

pmap.c 中 pages 定义为 page 的指针，即为一个数组的头指针，存放物理内存。

pmap.h 中，以下函数计算出页面的物理地址。

```
static inline u_long
    page2ppn(struct Page *pp)
    {
        return pp - pages;
    }
    /* Get the physical address of Page 'pp'.
    */
    static inline u_long
    page2pa(struct Page *pp)
    {
        return page2ppn(pp) << PGSHIFT;
    }
```

Thinking 2.6

请阅读 include/queue.h 以及 include/pmap.h, 将 Page_list 的结构梳理清楚,选择正确的展开结构(请注意指针)。

C

Thinking 2.7

在 mmu.h 中定义了 bzero(void *b, size_t) 这样一个函数,请你思考，此处的 b 指针是一个物理地址， 还是一个虚拟地址呢？

虚拟地址，由 init.c 中 bzero 的定义可知，对指针 b 指向的内容直接赋值，所以是虚拟地址。

Thinking 2.8

了解了二级页表页目录自映射的原理之后，我们知道，Win2k 内核的虚存管理也是采用了二级页表的形式，其页表所占的 4M 空间对应的虚存起始地址为 0xC0000000，那么，它的页目录的起始地址是多少呢？

0xC0000000 >> 22 <<12 = 0xC0300000

Thinking 2.9

注意到页表在进程地址空间中连续存放，并线性映射到整个地址空间，思考：是否可以由虚拟地址直接得到对应页表项的虚拟地址？上一节末尾所述转换过程中，第一步查页目录有必要吗，为什么？

可以使用一级页表，由虚拟地址直接得到对应页表项的虚拟地址。

有必要，节约页表内存，并且可以设置有效位。

Thinking 2.10 观察给出的代码可以发现，page_insert 会默认为页面设置 PTE_V 的权限。请问，你认为是否应该将 PTE_R 也作为默认权限？并说明理由

不应该，有 perm 参数来确定权限

Thinking 2.11

思考一下 tlb_out 汇编函数，结合代码阐述一下跳转到 NOFOUND 的流程？从 MIPS 手册中查找 tlbp 和 tlbwi 指令，明确其用途，并解释为何第 10 行处指令后有 4 条 nop 指令。

tlbp 查询 CP0_ENTRYHI 中虚拟地址是否存在 TLB 中：如果有则把匹配项的 index 保存到 Index 寄存器中；没有匹配则置 Index 的最高位为 1。、

tlwi 更新 TLB

nop 是为了使流水线暂停。

Thinking 2.12

显然，运行后结果与我们预期的不符，va 值为 0x88888，相应的 pa 中的值为 0。这说明我们的代码中存在问题，请你仔细思考我们的访存模型，指出问题所在。

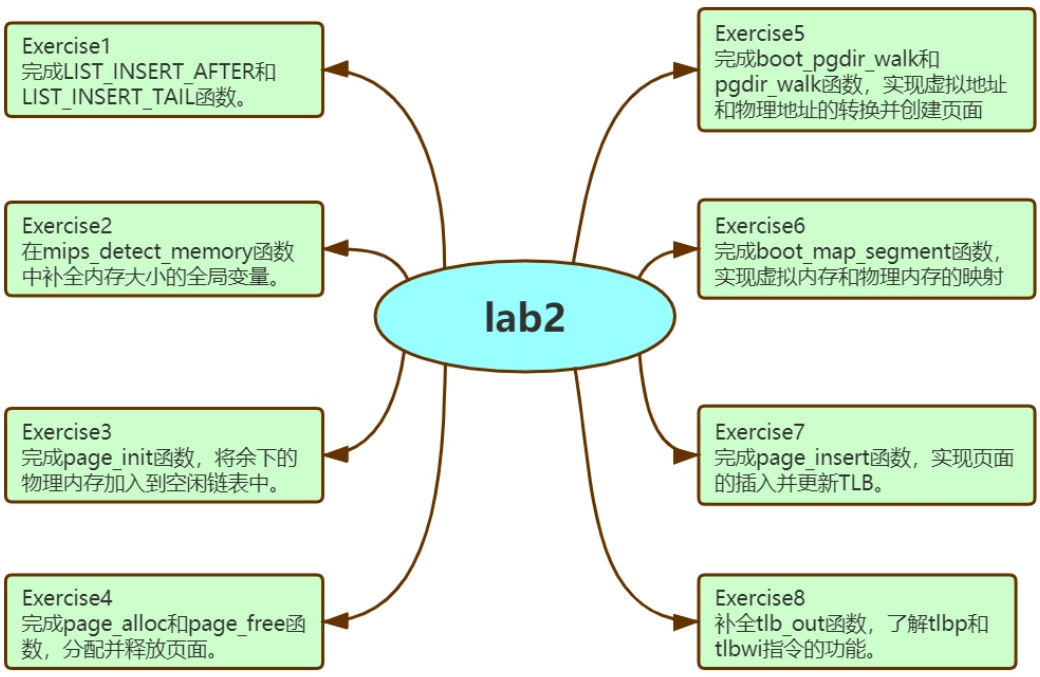
使用 va2pa()只是获得了 va 对应物理内存中的页的首地址，并未获得实际的 va 对应的 pa 地址，没有考虑页内偏移量。

Thinking 2.13

在 X86 体系结构下的操作系统，有一个特殊的寄存器 CR4，在其中有一个 PSE 位，当该位设为 1 时将开启 4MB 大物理页面模式，请查阅相关资料，说明当 PSE 开启时的页表组织形式与我们当前的页表组织形式的区别。

当开启 PSE 时，页的划分与当前基本一致，但在一级页表中增加使用了一位用于标识，用于区分从一级页表中寻找得到的地址是二级页表的入口地址还是 4MB 大小的页面地址。由于一级页表中一项能够映射到的地址本就是 4MB，所以整体结构变化不大。

二、实验难点图示

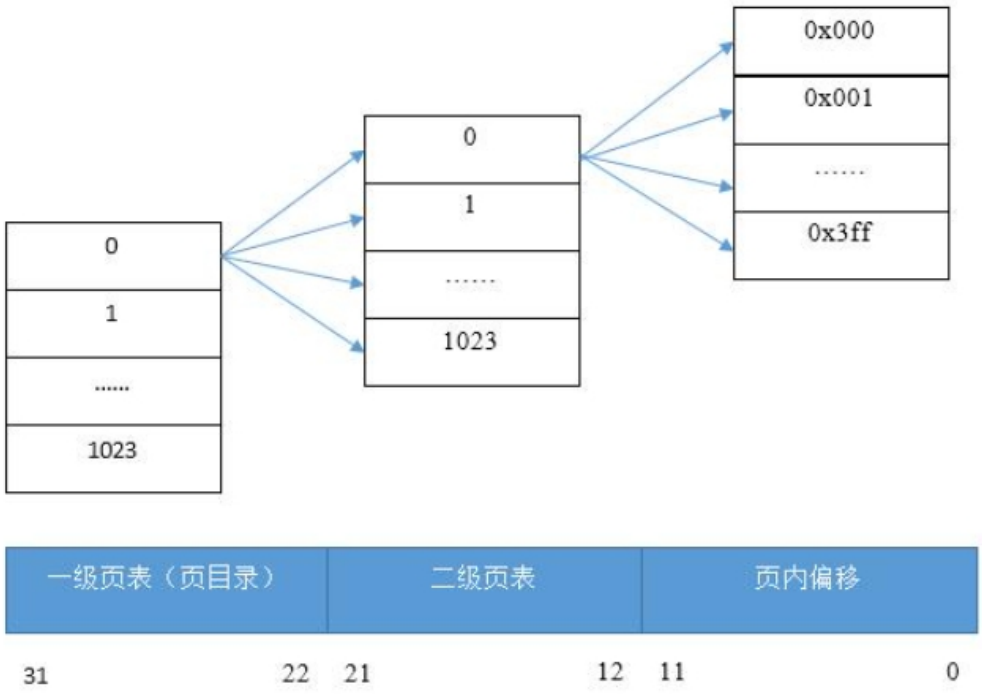


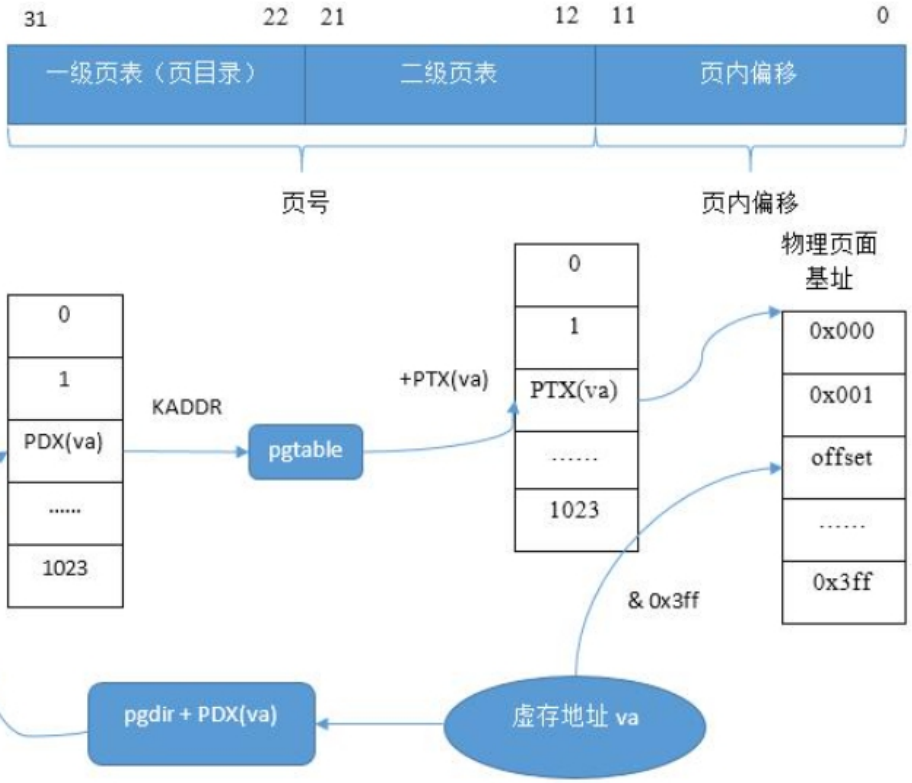
本次实验中的难点是对各种宏定义以及函数的理解的使用，例如 ROUND、LIST_HEAD、page2va、page2pa 等，如果不使用已有的函数，会使得各种操作、变换非常繁琐，而且不易理解，但如果使用这些函数，就需要阅读大量的实验代码，而不仅仅只是完成函数的填空，阅读并理解实验代码中的宏定义以及函数是本次实验的难点。

具体难点如下：

1.利用链表管理物理页面，链表结构体的定义与一般的链表有所不同，指针域中还存放了前一个节点指向后一个节点的指针的地址。而且利用了宏定义对链表进行操作，在编写宏定义时需要注意对于指针的使用。

2.boot_pgdir_walk 函数和 pgdir_walk 函数需要彻底理解页式内存管理的机制，尤其是二级页表的地址转换的机制。





3.对于页目录自映射的理解。

三、体会与感想

本次实验难度较 lab0、lab1 有了一定的提升，但随着学习的不断深入，难度的提升是必然的，我认为本次实验难度衔接十分合理，有助于我们接下来的学习。本次实验前前后后我大概花了 4 个下午 4 个晚上，加起来有 10 多个小时，但我认为这些时间的投入是十分必要的，从刚开始看教程时非常混乱到后来建立起比较清晰的内存布局，而且实验与理论课在进度上重合，两者可以相互促进。

四、指导书反馈

有些地方实验指导书与代码的注释有冲突，例如返回-E_NO_MEM 时，指导书上仅提到了 E_NO_MEM，虽然错误码返回负值是默认的行为，但对于刚刚接触操作系统代码的我们，这些不一致之处有时还是会带来不小的困惑，希望指导书和注释能够更加同步一些。

五、残留难点

对页式内存管理还不太熟练，尤其是页目录自映射这一部分，在上机时带来了很大的障碍，还需要多加学习。