Lab2 实验报告

15061200 李明轩

一，思考题

> Thinking 2.1 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？
>
> ■

答：由于在 C 语言中宏函数也只是单纯的字符串替换，因此如果不加一个循环体表示这只循环一次的话，有可能使得替换后的宏函数代码和上下文产生交互影响，比如在 LISTFOREACH 函数中就有一个 for 循环，我如果不给这个宏函数加上 while{0}那么根据缩进，替换后的部分都会被放在循环内部计算，如下图

```
LIST_FOREACH(page_temp, &page_free_list, pp_link)
    length++;
```

但有的时候我们希望宏函数不和下文代码产生联动，为了避免未知错误，我们就需要给宏函数套上 do-while{0}

> Thinking 2.2 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，给出你的想法。 ■

答：首先我们目前分配的内存主要分为两部分，一部分属于 kuseg 一部分属于 kseg，前者在虚拟地址到实际地址的映射过程中需要 mmu 转换，后者的内核虚拟地址可以直接由 CPU 进行处理映射到物理地址。因此 Page 结构体中存储的只是该页的虚拟地址，我们可以根据对应转换规则获得物理地址。

对于 kseg 区域的虚拟地址比如页目录的地址，可以发现一开始 alloc 返回的其实是内核虚拟地址

```
pgdir = alloc(BY2PG, BY2PG, 1);
```

接下来在访问页目录获得二级页表的时候我们直接通过内核虚拟地址获得了页目录项的值（也就是二级页表入口的物理地址）这一步虚拟地址到物理地址的映射是由 CPU 自动完成的。

```
pgdir_entryp = &pgdir[PDX(va)];
```

但是对于从空闲链表中分配出去的页面来说，需要经过 page2pa()函数的处理

```
*pgtable_entry = (page2pa(pp) | PERM);
```

在对应的页表中重新添加了虚拟地址到物理地址的映射（page2pa 内部还计算了物理页框号，ppn 并进行了左移）

> Thinking 2.3 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？ ■

答：page_alloc 函数中调用了该清空指定页面内容的函数，由此可见此处使用的是内核虚拟地址。但是由于是 kesg0 区域的虚拟地址，因此 CPU 可以直接对此部分虚拟地址映射到物理地址，反映到代码中就是可以直接使用*kva = 0 来清空。

```
bzero(page2kva(ppage_temp), BY2PG);
```

> Thinking 2.4　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？ ■

答：0xC0000000 对应第（0xC0000000>>12）个页表，每条页表项大小为 4B，因此页目录所在的页表项相对于页表起始地址的偏移为（0xC0000000>>12）*4　= 0x30 0000
故而页目录起始地址为 0xC030 0000

> Thinking 2.5　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？ ■

答：首先前两条函数只是用来暂存数据使用 `10　　tlbp` 才是整个流程的关键，查询 MIPS 手册如下，

The instruction:

```
tlbp     # TLB lookup
```

searches the TLB for an entry whose virtual page number and ASID matches those currently in **EntryHi** and stores the index of that entry in the **Index** register. Bit 31 of **Index** is set if nothing matches—this makes the value look negative, which is easy to test.

由此可见，该指令检查了 EntryHI 寄存器中的 va 是否在 TLB 中有记录，如果有则将 Index 寄存器的第三十一位置 0 否则置 1，这样当快表中不存在查询的页表项的时候，检查 Index 寄存器是否大于 0 即可，于是便有了接下来的 bltz

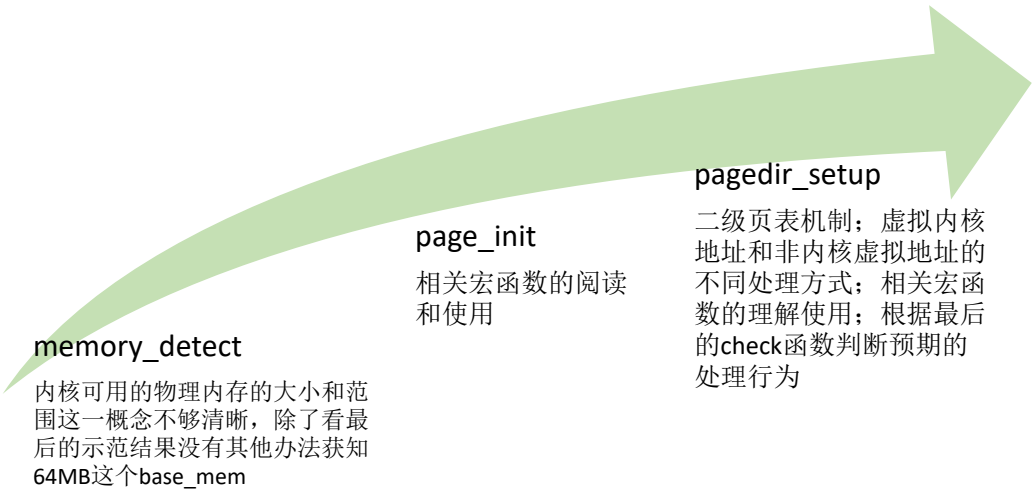
```
mfc0        k0,CP0_INDEX
bltz        k0,NOFOUND
```

只要 k0 寄存器中存储的 Index 值小于零，就跳转到 NOFOUND

> Thinking 2.6　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。 ■

答：在函数中访问某个地址的数值，虽然我们起的名字叫 pa 但是程序在处理的时候还是把 pa 的数值当作虚拟地址处理了，所以我们打印出来的结果相当于是 pa 作为虚拟地址映射到物理地址空间中的数值（如果没有设置过，结果就会是 0）

二，实验难点图示



三，体会与感想

本次 lab2 时间上没有安排好，导致后期赶进度而且有的地方理解不够透彻，一直到后来写本报告做思考题的同时才逐渐理解了以前的困惑。最大的感受在于做 OS 探索性问题的时候需要多多讨论，集思广益之下很多难点迎刃而解。

总体来看，做完 lab2 之后对于内存管理从课堂到实践有了更深层次的理解，尤其是两级页表机制，课本上通俗易懂，真正的实现却颇有带着枷锁跳舞之感，页目录遍历过程中才产生了页表。另外整个实验代码中最让人兴奋的是各种精巧的宏函数，未曾想到宏定义可以通过换行和简单的字符串替换实现如此复杂的功能，而一个个宏函数组合在一起可以实现操作系统如此底层的功能。同时我也发现，既然越接近底层，出于资源等各种限制，设计人员倾向于将程序设计的越发精巧，而到了高层资源多了，大部分程序员也就不重视程序的优化了。我觉得这种懒于思考的态度是我们以后学习工作必须避免的。

四，指导书反馈

实验代码的变量命名上应该再优化一下，比如 extern char[] end（找不到，不知道是谁的 end），freememory（指针还是一块内存数组） 这样模棱两可的命名，给大家的实验平添难度。