Lab 2 实验报告

潘文峥（学号：520030910232）

思考题1：

优势：相比单级页表，多级页表能节省页表占用的空间。尤其在物理地址空间存在大量空洞时，多级页表通过离散的空间分配，避免单级页表连续的预分配，能有效减少不必要的页表存储开销。

劣势：完成一次地址转换需要多次访问页表，可能带来更多的时间开销。

4KB粒度：0~4GB地址范围对应物理页数：4GB/4KB = 2^{20}，每个页表页的页表项数：4KB/64bit = 512 = 2^9，所需三级页表数：$2^{20}/2^9 = 2^{11} = 2048$，所需二级页表数：$2048/512 = 4$，加上零级页表和一级页表各一张，页表页总数为`2054`张（占用大小8216KB）。

2MB粒度：0~4GB地址范围对应物理大页数：4GB/2MB = 2^{11}，二级页表的大页项数：512，所需二级页表数：$2^{11}/512 = 4$ 页，加上零级页表和一级页表各一张，页表页总数为`6`张（占用大小24KB）。

练习题2：

实现思路：

首先设置零级页表和一级页表，其页表项分别存储下一级页表的起始地址：

```
boot_ttbr1_l0[GET_L0_INDEX(kva)] = ((u64) boot_ttbr1_l1) | IS_TABLE | IS_VALID;
boot_ttbr1_l1[GET_L1_INDEX(kva)] = ((u64) boot_ttbr1_l2) | IS_TABLE | IS_VALID;
```

对于二级页表，高地址页表起始地址为`KERNEL_VADDR`，相应的物理内存和设备地址结束处对应页号分别为`GET_L2_INDEX(kva) + PERIPHERAL_BASE / SIZE_2M`和`GET_L2_INDEX(kva) + PHYSMEM_END / SIZE_2M`，和低地址页表配置类似，以2M为粒度，利用`for`循环设置二级页表的每一项指向一个2M物理页。

思考题3：

在`el1_mmu_activate`函数中使能MMU后，也就是寄存器`SCTLTR_EL1`设置完成后，ChCore使用的都是虚拟地址，而此时下一条指令地址仍位于低地址空间，因此需要配置低地址页表。

经验证，将低地址页表配置代码注释后，`el1_mmu_activate`函数不能正常运行结束，ChCore不能正常跳转到高地址继续其余初始化工作，与预期一致。（下图分别为不配置/配置低地址页表的运行情况）

```
boot: init_c
[BOOT] Install boot page table
```

```
boot: init_c
[BOOT] Install boot page table
[BOOT] Enable el1 MMU
[BOOT] Jump to kernel main
```

练习题4：

实现思路：

`split_page`：传入page的order与目标order相等时，直接返回该page；否则page->order减一，并将其伙伴块放入对应阶的空闲链表，最后递归地调用`split_page`函数继续分裂当前块。

`buddy_get_pages`：找到目标order大小的空闲链表，如果有空闲块则取出一块分配；否则依次尝试直到找到空闲的更大块，调用`split_page`函数得到一个目标order的块进行分配。被分配的块从对应阶的空闲链表中移除。

`merge_page`：如果当前块阶数等于上限阶数（`BUDDY_MAX_ORDER`），或当前块的伙伴块不空闲（`allocated`为1），则直接返回当前块；否则，page->order加一，`allocated`置零，并递归地调用`merge_page`函数尝试继续合并当前块。

`buddy_free_pages`：将当前块`allocated`置零表示未分配，调用`merge_page`函数尝试进行合并操作，将得到的新空闲块加入对应阶的空闲链表。

练习题5：

实现思路：

首先认真分析各函数的参数以及宏定义，明确各指针变量所指向的内容和意义。

`query_in_pgtbl`：该函数的功能是，传入页表起始地址、一个虚拟地址变量（`va`）和一个待填充的物理地址指针（`*pa`），在页表中查询，如果存在合法映射，则在传入的`*pa`指针所指位置填入该虚拟地址对应的物理地址并返回`0`，否则返回`-ENOMAPPING`。首先，循环调用三次`get_next_ptp(cur_ptp, i, va, &next_ptp, entry, false)`函数，依次取该虚拟地址对应的各级页表索引，如果遇到未映射，直接返回`-ENOMAPPING`，如果正常映射，则将最后得到的**物理页号**加上**偏移量**（`va & 0xfff`）算出**物理地址**，赋值给`*pa`，并返回`0`。

`map_range_in_pgtbl`：该函数传入一个虚拟地址`va`、一个物理地址`pa`和一个范围长度`len`，功能是在页表中填写`[pa, pa+len)`范围的地址映射。使用两层循环，外层遍历范围长度`len`，内层循环调用三次`get_next_ptp(cur_ptp, i, va, &next_ptp, entry, true)`函数，此时，遇到未映射的页表项将添加映射，最后将物理页号写入三级页表项。

`unmap_range_in_pgtbl`：该函数传入一个虚拟地址`va`、一个物理地址`pa`和一个范围长度`len`，功能是在页表中取消`[pa, pa+len)`范围的地址映射。类似地，使用两层循环，外层遍历范围长度`len`，内层循环调用两次`get_next_ptp(cur_ptp, i, va, &next_ptp, entry, false)`函数，将最后一级页表项的`is_valid`位置零表示映射被取消。

练习题6：

实现思路：

`map_range_in_pgtbl_huge`：总体思路与`map_range_in_pgtbl`类似，根据传入的待映射范围，依次尝试分配1G大页、2M大页和4K页。对大页的分配需要在对应页表页的`pfn`字段填入对应物理大页的索引，并将`is_page`标志位置零表示所指是大页而非下一页标的地址。

```
(next_ptp->ent)[GET_L2_INDEX(va)].l3_page.pfn = pa >> 12;
(next_ptp->ent)[GET_L2_INDEX(va)].l3_page.is_page = 0;
```

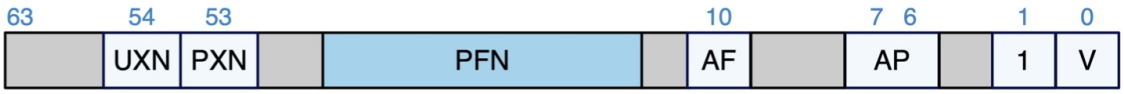
同时，`query_in_pgtbl`函数也增加对大页查询的支持，计算物理地址时将分别用到以下三种偏移量。

```
offset_4K = va & 0xfff;
offset_2M = (GET_L3_INDEX(va) << 12) + offset_4K;
offset_1G = (GET_L2_INDEX(va) << (12 + 9)) + offset_2M;
```

`unmap_range_in_pgtbl_huge`：总体思路与`unmap_range_in_pgtbl`类似，根据传入的待映射范围，依次尝试取消映射1G大页、2M大页和4K页，将最后一级页表项的`is_valid`位置零表示映射被取消。

思考题7：

如下图所示，三级页表项的`AP`字段描述了该物理页的读写权限。为支持写时拷贝，需要将`AP`字段置为`11`，即**只读**权限，一旦某应用程序对该区域进行修改，就会触发访问权限异常，此时，操作系统会将异常的物理页拷贝一份，并把`AP`字段置为`01`（**可读可写**），再映射给该应用进程，恢复进程执行。



页描述符：指向4K页

思考题8：

粗粒度映射一次性分配2M大页可能在后续运行中造成较多空间浪费。

挑战题9：

利用练习题5中实现的页表映射函数，简化了页表填写过程，两次调用`map_range_in_pgtbl(ttbr1_l0, va, pa, len, flags)`函数，分别以4KB粒度映射`PHYSMEM_START ~ PERIPHERAL_BASE`和`PERIPHERAL_BASE ~ PHYSMEM_END`范围的物理地址到内核页表中。

实验结果：

按要求完成代码填写，`make grade` 评分`100/100`：