华东师范大学计算机科学与技术学院上机实践报告

课程名称：操作系统	**年级**：2022 级	**上机实践日期**：2024 年 5 月 7 日
指导教师：李东	**姓名**：岳锦鹏	**学号**：10213903403
实验名称：实验三　虚拟内存管理		

一、 研读理解相关代码后，回答以下问题：

1、 uCore 中的段页式存储管理方案与理论课中讲述的方案有何不同?

μCore 中简化了分段机制，将逻辑地址直接恒等映射到线性地址，之后使用分页机制将线性地址通过分页映射到物理地址。由于使用了分页机制，因此需要对空闲物理块管理，对空闲物理块的管理使用了链表来管理，链表按照地址排序。

跟我们理论课中的“段页式存储管理”和“使用链表管理存储空间”的方案相似，不同之处在于分段机制简化了，并且只使用链表管理了空闲物理块（已分配的物理块在页表中有记录，所以不需要管理）。

2、 试简述 uCore 中缺页异常的处理过程，它与理论课中讲述的过程有何不同?

在 vmm.c 的注释中有这样一行：

```
* CALL GRAPH: trap--> trap_dispatch-->pgfault_handler-->do_pgfault
```

显然，在 μCore 中，缺页异常从 `trap` 触发，之后通过 `trap_dispatch` 根据中断号（`tf->tf_trapno`）分发给不同的处理程序，这里是缺页中断（`T_PGFLT`），所以分发给 `pgfault_handler`，再调用 `do_pgfault`，之后在 `do_pgfault` 中找到一个 pte，并且分配内存或者从交换分区换入一个页面，或者由于访问权限不正确而直接返回失败。

与理论课中讲述的过程的不同是进行了简化，取消了快表，也就是只有一级索引而没有二级索引。

3、 为了支持页面置换，在数据结构 Page 中添加了哪些字段，其作用是什么?

```
struct Page {
    int ref;                        // page frame's reference counter
    uint32_t flags;                 // array of flags that describe the status of the
    ↪  page frame
    unsigned int property;          // the num of free block, used in first fit pm
    ↪  manager
    list_entry_t page_link;         // free list link
    list_entry_t pra_page_link;     // used for pra (page replace algorithm)
    uintptr_t pra_vaddr;            // used for pra (page replace algorithm)
};
```

添加了 pra_page_link 和 pra_vaddr 这两个字段，pra_page_link 是用来将物理块组织成 FIFO 队列（使用链表实现）用来进行页面置换的，pra_vaddr 是物理块对应的虚拟地址，用来记录访问哪个虚拟地址的时候产生的缺页，以便进行页面置换。

4、 请描述页目录项页 (PDE) 和页表目录项 (PTE) 的组成部分对 uCore 实现页面置换算法的潜在用处。请问如何区分未映射的页表表项与被换出页的页表表项？请描述被换出页的页表表项的组成结构。

用处在于记录某个页面在内存中还是在外存对换区中，以及对应物理地址（在内存中）或者 offset（在外存对换区中），在下次页面换入或换出时确保页面被放到正确的位置上。

由于未映射的页表表项全是 0，所以被换出的页的 offset 从 1 开始以区分。这样如果页表表项全是 0 就说明是未映射，如果在 offset（高 24 位）中存在 1，那么就是被换出的。

被换出页的页表表项复用了 pte 的结构，在 memlayout.h 中有这样一行：

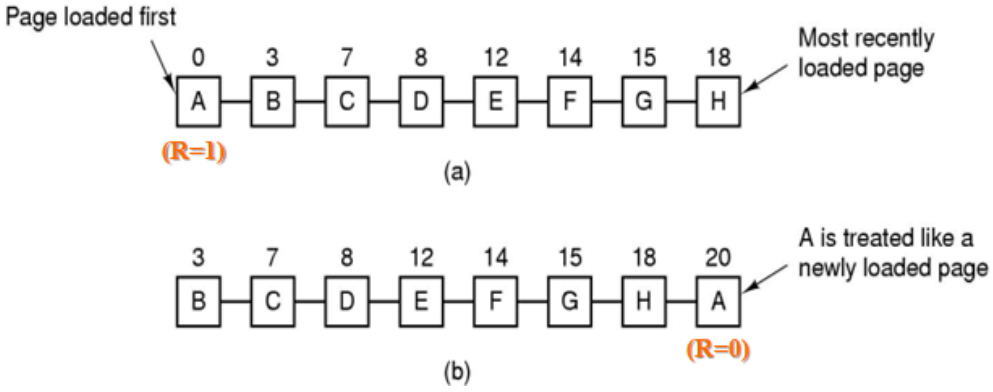
```
typedef pte_t swap_entry_t; //the pte can also be a swap entry
```

再根据示意图：

可以看到其高 24 位为 offset，代表在外存对换区里的位置；存在位（最低位）为 0，表示此页面不存在对应的物理块；中间的 7 位都保留暂不使用。

二、 程序设计与实现的基本思路

1、 在 μCore 中原先实现的是 FIFO 页面置换算法，这里要改成第二次机会页面置换算法，可以发现页面的组织方式不需要更改，仍然使用链表，每次插入页面时仍然是添加到链表头部不需要更改，只需要更改换出页面的算法即可。



2、 课上讲的这张示意图，右侧是链表的头部，左侧是链表的尾部，每次插入的页面在最右侧即链表的头部。当需要换出页面时，从左侧（链表尾部）取出一个页面，如果这个页面的访问位是 1，那么将其访问位改为 0，并放到右侧（链表头部）。接着再从左侧（链表尾部）检测下一个页面，直到找到一个访问位为 0 页面的作为换出的页面。所以可以写出代码如下：

```
// 从后往前，最后面的是最早访问的
list_entry_t *current = head->prev;
list_entry_t *le = NULL;
for (; current != head; current = current->prev) {
    pte_t *ptep = get_pte(mm->pgdir, le2vma(current, vm_start), 0);
    if (*ptep & PTE_A) { // 访问位为 1
        *ptep &= !PTE_A; // 清除访问位
        list_del(current); // 从链表中移除
        list_add(head, current); // 添加到链表头部
    } else {
        le = current;
        break;
    }
}
```

le 即为最终的找到的用来换出的页面。

3、 当然，还需要完善一些问题。当页面只有一个并且访问位为 1 时，从链表中移除此页面再放入后链表结构并未改变，所以下一次的 current 指向链表头结点，循环退出了，但此时 le 还是空的！但期望的返回结果应该是这一个页面，所以需要对这种情况进行处理：

```
if (le == NULL) {
    le = head->prev; // 循环一遍找不到就最后一个
}
```

当循环一遍仍然找不到可以换出的页面的时候就选择最后一个页面（和 FIFO 一样）作为换出的页面。当然，其实使用第二次机会页面置换算法的话，页面大于一个的时候必定能找到一个换出的页面的（请自行验证），而且也不会出现页面为 0 个情况（总不能页表数量为 0 吧），所以其实这样就只是解决了页面只有一个时的异常。

4、 测试代码 _fifo_check_swap 仍然需要修改，因为使用第二次机会页面置换算法，缺页次数肯定与 FIFO 算法不尽相同。测试代码中的 pgfault_num 即为缺页次数。

5、 由于时间紧迫，一些细节可能没有考虑周到，如有发现错误欢迎在下方的链接中提 Issue。

三、 代码

1、 https://gitea.shuishan.net.cn/10213903403/os_kernel_lab

2、 也可以看上传的附件。