Lab2-Extra-Swap

题目背景

在理论课程中，我们学习了**交换**技术。它实现进程在内存与外存之间的交换，因而获得更多的虚拟内存空间。

简单来说，交换空间（swap）是外存上的一块区域，当系统物理内存不足时，内核会将内存中不常访问的数据保存到 swap 上，这样系统就有更多的物理内存为各个进程服务，而当系统需要访问 swap 上存储的内容时，再将 swap 上的数据加载到内存中。这样相当于我们获得了更多的虚拟存储（通过使用一部分外存）。

在本题中，我们会实现一个较为简单的交换机制，使得在没有空闲的物理页面时，可以暂时将正在使用的一页内存换出，同时释放出一页物理页面用于使用。

题目描述

我们建立的交换机制可以分为两部分，“换入”部分，以及“换出”部分。

当我们没有空闲的物理页面时，我们进行“换出”，即申请物理页面时，如果没有可用的页面，我们换出一页正在使用的物理页，供申请者使用。

当我们需要访问某个 kuseg 段的虚拟地址时，我们会检查这个虚拟地址对应的虚拟页是否已被换出到外存，如果是，则我们将其“换入”。

> 虚拟页被换入的物理页可能与其被换出时不同，但需要保证换入后**物理页中的数据**以及**页表项中的权限位**与换出时相同。为此，我们需要在换出时利用外存来保存数据。

题目要求

在本题中，你需要使用物理地址属于 `[0x3900000, 0x3910000)` 的这 16 个物理页以及外存来实现“交换”。

- 在本题中我们把这 16 个物理页叫做**可交换**的物理页。
- 为了区分这些**可交换**的物理页，我们建立了一个新的**空闲可交换页面链表** `page_free_swapable_list`。

同时，我们将提供部分代码（请参看**实验提供代码**部分），你需要将其粘贴至 `kern/pmap.c` 之后，并补全或者实现如下几个函数：

换出部分（`struct Page *swap_alloc(Pde *pgdir, u_int asid)`）

本函数的功能为：

- 当存在**空闲**且**可交换**的物理页（`page_free_swapable_list` 链表非空），只需从 `page_free_swapable_list` 中取出头部并返回。
- 若不存在**空闲**且**可交换**的物理页（`page_free_swapable_list` 链表为空），需要从 `[0x3900000, 0x3910000)` 中选取一个物理页，将其**换出**到外存，并将其返回。
    - 本题不限制页面置换的策略，也就是说，你可以使用任意策略来选取一个物理页，将其换出到外存。

注意：

- 实验提供代码中的 `swap_init` 函数将 `[0x3900000, 0x3910000)` 对应的 `Page` 结构体从 `page_free_list` 中移除并插入到 `page_free_swapable_list` 中。因此，`swap_alloc` 所返回的 `Page` 对应的物理页，其物理地址必须是处于 `[0x3900000, 0x3910000)` 中的。
- 我们**保证**：在每次测试中，传入的 `pgdir` 和 `asid` 是**唯一**的。

换入部分（`void swap_lookup(Pde *pgdir, u_int asid, u_long va)`）

本函数的功能为：

- 当地址空间 `asid` 中的虚拟地址 `va` 在页目录 `pgdir` 中存在映射，但对应物理页面被换出时，调用 `swap` 函数将其换入
- 调用 `page_lookup` 函数，返回 `va` 对应的页表项

注意：

- 我们保证：在每次测试中，传入的 `pgdir` 和 `asid` 是唯一的
- **传入的 va 不一定是页对齐的。**

解释：

本函数的实现已经给出，你需要实现该函数中调用的 `swap` 函数和 `is_swapped` 函数。

- `int is_swapped(Pde *pgdir, u_long va)`
  - 本函数的功能为：当虚拟地址 `va` 在页目录 `pgdir` 中存在映射且对应物理页面被换出时，返回非 0 值，否则返回 0。
- `void swap(Pde *pgdir, u_int asid, u_long va)`
  - 本函数的调用者需保证虚拟地址 `va` 映射到的物理页已被换出到外存。
  - 本函数的具体功能为：将页目录 `pgdir` 中虚拟地址 `va` 映射的物理页从外存中换入内存，并且更新其对应的页表项。换入时需要使用 `swap_alloc` 来申请一个物理页。其中 `asid` 参数用于传递给 `swap_alloc` 函数、更新页表时无效化对应的 TLB 表项。
- **外存模拟部分**

  由于还没有学习如何访问外存，我们使用一个数组 `swap_disk` 来模拟外存（大小为 64 个物理页大小）。

  我们使用如下两个接口函数来申请、释放外存空间：
  - `u_char *disk_alloc()`
    - 申请一页大小的外存空间（页对齐），返回值为这片空间的起始地址。外存空间的一页大小为 4096 字节，与内存中的页大小一致。
      - 返回的地址为 kseg0 段的，指向 `swap_disk` 数组内空间的地址。
  - `void disk_free(u_char* da)`
    - 释放 `da` 起始的一页外存空间。

设计提示

我们给出一种可行的设计，当然，你也可以略过本节自己进行设计。

当没有空闲的物理页时，我们需要进行换出操作。在本设计中，我们在页表项中增加了一个新的标志位 `PTE_SWP`（在下发的头文件 `swap.h` 中已有定义）。

- 当 `PTE_SWP` 为 `1` 且 `PTE_V` 为 `0` 时:
  - 对应的虚拟地址映射到的物理内存**有效但被换出**，实际的内容存在外存上，该页表项的高 20 位为内容在外存上的外存页号。
- 软件应保证不会出现 `PTE_SWP` 为 `1` 且 `PTE_V` 为 `1` 的页表项。
- 当 `PTE_SWP` 为 `0` 时，页表项的含义与 Lab2 课下定义的相同。
- 我们可以通过 `da / BY2PG` 计算 `da` 对应的外存页号

当我们希望将某个虚拟地址对应的物理页从外存中换入内存时：

1. 使用 `swap_alloc` 申请一个物理页 `p`
2. 将外存中以 `da` 起始的一页内容拷贝到该物理页 `p` 上（`da` 为换出时内容在外存上的地址）
3. 对指定页表中，所有“`PTE_SWP` 为 `1` 且 `PTE_V` 为 `0` 且高 20 位为 `da` 对应的外存页号”的页表项，做如下操作：
   1. 将 `PTE_V` 置 `1`
   2. 将 `PTE_SWP` 置 `0`
   3. 在高 20 位中填入 `p` 对应的物理页号
   4. 维持其它权限位不变
   5. 无效化旧 TLB 表项
4. 使用 `disk_free` 释放 `da` 起始的一页外存空间

当我们需要换出一个内存中的物理页至外存时：

1. 从 `[0x3900000, 0x3910000)` 的内存空间中，选择一个物理页 `p`
2. 使用 `disk_alloc` 申请一页大小的外存空间，记该外存空间的起始地址为 `da`
3. 对指定页表中，所有 `PTE_V` 为 `1` 且高 20 位为 `p` 的物理页号的页表项，做如下操作：
   1. 将 `PTE_V` 置 `0`
   2. 将 `PTE_SWP` 置 `1`
   3. 在高 20 位中填入 `da` 对应的外存页号
   4. 维持其它权限位不变
   5. 无效化旧 TLB 表项
4. 将物理页 `p` 上的内容拷贝到外存中 `da` 起始的一页空间上
5. 释放物理页 `p`，也就是将其插回 `page_free_swapable_list` 链表中

任务总结

在提交前，你需要完成以下任务：

1. 换入部分：

   完成 `is_swapped` 函数。

   完成 `swap` 函数，维护 `page_free_swapable_list` 链表，适时无效化 TLB 中的旧表项。

2. 换出部分：

   完成 `swap_alloc` 函数，维护 `page_free_swapable_list` 链表，适时无效化 TLB 中的旧表项。

实验提供代码

请将本部分提供代码附加在你的 kern/pmap.c 的尾部，然后开始做题。

```
#include <swap.h>
struct Page_list page_free_swapable_list;
static u_char *disk_alloc();
static void disk_free(u_char *pdisk);

void swap_init() {
  LIST_INIT(&page_free_swapable_list);
  for (int i = SWAP_PAGE_BASE; i < SWAP_PAGE_END; i += BY2PG) {
    struct Page *pp = pa2page(i);
    LIST_REMOVE(pp, pp_link);
    LIST_INSERT_HEAD(&page_free_swapable_list, pp, pp_link);
  }
}

// Interface for 'Passive Swap Out'
struct Page *swap_alloc(Pde *pgdir, u_int asid) {
  // Step 1: Ensure free page
  if (LIST_EMPTY(&page_free_swapable_list)) {
    /* Your Code Here (1/3) */

  }

  // Step 2: Get a free page and clear it
  struct Page *pp = LIST_FIRST(&page_free_swapable_list);
  LIST_REMOVE(pp, pp_link);
  memset((void *)page2kva(pp), 0, BY2PG);

  return pp;
}

// Interfaces for 'Active Swap In'
static int is_swapped(Pde *pgdir, u_long va) {
  /* Your Code Here (2/3) */

}

static void swap(Pde *pgdir, u_int asid, u_long va) {
  /* Your Code Here (3/3) */

}

Pte swap_lookup(Pde *pgdir, u_int asid, u_long va) {
  // Step 1: If corresponding page is swapped out, swap it in
  if (is_swapped(pgdir, va)) {
    swap(pgdir, asid, va);
  }

  // Step 2: Look up page table element.
  Pte *ppte;
  page_lookup(pgdir, va, &ppte);
```

```
  // Step 3: Return
  return ppte == NULL ? 0 : *ppte;
}

// Disk Simulation (Do not modify)
u_char swap_disk[SWAP_DISK_NPAGE * BY2PG] __attribute__((aligned(BY2PG)));
u_char swap_disk_used[SWAP_DISK_NPAGE];

static u_char *disk_alloc() {
  int alloc = 0;
  for (;alloc < SWAP_DISK_NPAGE && swap_disk_used[alloc]; alloc++) {
    ;
  }
  assert(alloc < SWAP_DISK_NPAGE);
  swap_disk_used[alloc] = 1;
  return &swap_disk[alloc * BY2PG];
}

static void disk_free(u_char *pdisk) {
  int offset = pdisk - swap_disk;
  assert(offset % BY2PG == 0);
  swap_disk_used[offset / BY2PG] = 0;
}
```

本地测试说明

你可以使用：

- `make test lab=2_swap && make run` 在本地测试上述样例（调试模式）
- `MOS_PROFILE=release make test lab=2_swap && make run` 在本地测试上述样例（开启优化）

或者在 `init/init.c` 的 `mips_init` 函数中自行编写测试代码并使用 `make && make run` 测试。

在样例测试中，我们会申请 32 个页面并向其中写入一些内容，随后检查内容是否正确。

如果样例测试中输出了如下结果，说明你通过了本地测试。

```
Memory size: 65536 KiB, number of pages: 16384
to memory 80430000 for struct Pages.
pmap.c:  mips vm init success
Page Init Successed.
Swap Init Successed.
Swap Test Begin.
1x Page Used
2x Page Used
Congratulation!
```

评测说明

评测时使用的 mips_init() 函数示意如下：

```
void mips_init() {
  mips_detect_memory();
  mips_vm_init();
  page_init();
  swap_init();
  swap_test();
  halt();
}
```

- 保证不会出现外存空间不足的情况。
- 保证传入的页目录**不使用**可交换的物理页。
  - 对于二级页表的分配，请注意在 `page_insert` 函数中保持使用原有的 `page_alloc`，以避免将页表存储在可交换的物理页上。
- 在每次测试中，传入的 `pgdir` 和 `asid` 都是唯一的。
- 在 `swap_test` 函数中，测试程序的行为仅限于：
  - 使用 `swap_alloc` 函数分配一页物理页
  - 使用 `page_insert` 将物理页插入页表中
    - **注意**：我们保证对于每个虚拟地址 `va`，只会调用 `1` 次 `page_insert`
  - 调用 `swap_lookup` 函数获得某个 `va`（位于 kuseg 中）对应的页表项，将其填入 TLB，然后对 `va` 进行读写
  - 在页表中读页表项的值

具体要求和分数分布如下：

测试点序号	评测说明	分值
1	与本地测试相同	10
2	申请的物理页数目小于等于 16	10
3	申请较多的物理页	10
4	多次随机读写	10
5	多个虚拟地址映射到同一物理页	18
6	多个虚拟地址映射到同一物理页（规模更大）	19
7	多个虚拟地址映射到同一物理页、多次随机读写	20
8	极限负载情况（申请了79个物理页）	3