Project 3a: Virtual Memory

Preliminaries

> Fill in your name and email address.

Qizhi Chen chenqizhi@stu.pku.edu.cn

> If you have any preliminary comments on your submission, notes for the TAs, please give them here.

操作系统真是精妙，一个bug能调八个小时。

那些能够设计出复杂系统的人真是厉害，能够设计出pintos的人真是特别特别厉害。

> Please cite any offline or online sources you consulted while preparing your submission, other than the Pintos documentation, course text, lecture notes, and course staff.

Page Table Management

DATA STRUCTURES

> A1: Copy here the declaration of each new or changed struct or struct member, global or static variable, typedef, or enumeration. Identify the purpose of each in 25 words or less.

```
struct thread
  {
    struct hash sup_page_table;         /**< Supplemental page table. */
  };
```

For restore supplied page table.

```
struct suppl_pte_data{
  struct file * file;
  off_t ofs;
  uint32_t read_bytes;
  uint32_t zero_bytes;
  bool writable;
};

struct suppl_pte
{
  int type;
  bool is_loaded;
  void *vaddr;
  struct suppl_pte_data data;
  size_t swap_index;
  bool swap_writable;
  struct hash_elem elem;
```

```
};
```

the supplied page table entry

```
struct frame_table_entry{
  struct thread *owner;
  uint32_t *pte; //page_table_entry
  void *kaddr; // kernel address
  void *vaddr; // virtual address
  struct list_elem elem;
};
```

the frame table entry for restore all the frame.

ALGORITHMS

> A2: In a few paragraphs, describe your code for accessing the data stored in the SPT about a given page.

At first, alloc all the user's pool, use the frame_table_entry to restore them.

then, any palloc turns into get a frame from my restored structure.

When have a page_fault, by a hash_table (sup_page_table for each thread) , It can get his suppl_pte , if this page should exists but not loaded , alloc a frame (and set things like dirty bit).

I use a `clock algorithm` to choose whether frame should be swapped out.

After alloc, set the page_table (using install_page function) to set the page's virtual_address to be the kernel_address ,which can be access just with an offset.

Then, the next time i accessing the page_table, it should not be page_fault.

> A3: How does your code coordinate accessed and dirty bits between kernel and user virtual addresses that alias a single frame, or alternatively how do you avoid the issue?

I made the user can only access USER_POLL, that means this would not happen if kernel and user concurrently access the same page.

SYNCHRONIZATION

> A4: When two user processes both need a new frame at the same time, how are races avoided?

By adding a lock when entering `frame_alloc` or `frame_free`

which means only one user process can get a new frame at one time.

By the way, when releasing the resources when process_exit , I also add a lock so that no more eviction could happen with partly dealing frames.

RATIONALE

> A5: Why did you choose the data structure(s) that you did for representing virtual-to-physical mappings?

for sup_page_table, i can restore all things i need to complete things i need.

```
struct suppl_pte_data{
```

```
  struct file * file;
  off_t ofs;
  uint32_t read_bytes;
  uint32_t zero_bytes;
  bool writable;
};

struct suppl_pte
{
  int type;
  bool is_loaded;
  void *vaddr;
  struct suppl_pte_data data;
  size_t swap_index;
  bool swap_writable;
  struct hash_elem elem;
};
```

such like, type [FILE ? / STACK ? in SWAP ?] `is_loaded` is it in memory?

```
struct frame_table_entry{
  struct thread *owner;
  uint32_t *pte; //page_table_entry
  void *kaddr; // kernel address
  void *vaddr; // virtual address
  struct list_elem elem;
};
```

for frame table, kaddr and vaddr are essential, while pte is for accessing page_set's dirty / accessed bit.

Paging To And From Disk

DATA STRUCTURES

> B1: Copy here the declaration of each new or changed struct or struct member, global or static variable, typedef, or enumeration. Identify the purpose of each in 25 words or less.

```

struct suppl_pte
{
  size_t swap_index;
  bool swap_writable;
};
```

any suppl_pte restore the `swap_index` and `swap_writable`.

it means where the disk is and is this page writable

ALGORITHMS

> B2: When a frame is required but none is free, some frame must be evicted. Describe your code for choosing a frame to evict.

`clock_algorithm`

set the global_current_elem as the one pointer, each time if this page's accessed bit is 0, then choose it, or set it into 0 and goto the next list_element.

```
static struct frame_table_entry* find_frame_by_clock(){
    // printf("START_FINDING\n");
    struct frame_table_entry *vf;
    struct list_elem *e;
    for(;;){
        for(e = global_current_elem; e != list_end(&frame_table); e =
list_next(e)){
            vf = list_entry(e, struct frame_table_entry, elem);
            if(vf->owner == NULL ){//逆天，pagedir清空表示整个线程已经结束了
                global_current_elem = list_next(e);
                return vf;
            }
            if(vf -> owner != NULL && vf -> vaddr == NULL){
                continue;
            }
            if(!pagedir_is_accessed(vf->owner->pagedir, vf->vaddr)){
                global_current_elem = list_next(e);
                return vf;
            }
            pagedir_set_accessed(vf->owner->pagedir, vf->vaddr, false);
        }global_current_elem = list_begin(&frame_table);
    }
    return NULL;//IMPOSSIBLE
}
```

> B3: When a process P obtains a frame that was previously used by a process Q, how do you adjust the page table (and any other data structures) to reflect the frame Q no longer has?

```
static bool evict_frame(struct frame_table_entry *vf){
    struct thread *t = vf->owner;
    struct suppl_pte *spte = get_spte (&t->sup_page_table, vf->vaddr);
    ASSERT(spte != NULL);
    if(pagedir_is_dirty(t->pagedir, spte->vaddr) || spte->type !=
SPTE_TYPE_FILE){
        size_t swap_index = swap_out(vf -> kaddr);
        if(swap_index == SWAP_ERROR) PANIC("SWAP OUT FAILED");
        spte->type |= SPTE_TYPE_SWAP; //只要写过一次，永远都是SWAP了
        spte->swap_index = swap_index;
        spte->swap_writable = *(vf->pte) & PTE_W;
    }
    memset(vf->kaddr, 0, PGSIZE);
    spte->is_loaded = false;
    pagedir_clear_page(t->pagedir, spte->vaddr);
    return true;
}
```

set Q's supplied page table, to make it get into swap or just set it is not loaded(if a file only has read operation)

and use `pagedir_clear_page` to set it's done.

SYNCHRONIZATION

> B5: Explain the basics of your VM synchronization design. In particular, explain how it prevents deadlock. (Refer to the textbook for an explanation of the necessary conditions for deadlock.)

用了两个锁，一个叫做 `frame_lock`，一个叫做 `frame_evict_lock`，

在 `alloc_frame` 和 `free_frame` 的时候用了 `alloc_frame`，全部都用，整个都用。

在 `alloc_frame` 需要找到一个可以用的frame的时候用了 `frame_evict_lock` 。

在内存结束释放资源的时候，使用 `hash_destroy` 去清理内存，首先获取 `frame_evict_lock` ，在 `free` 的时候可能会获取 `frame_lock`。

注意到为了防止死锁，就是在 `alloc_frame` 必须先持有 `frame_evict_lock` 然后再持有 `frame_lock` 。这样就保证了任何时候如果没有`frame_evict_lock` 就不会死锁

> B6: A page fault in process P can cause another process Q's frame to be evicted. How do you ensure that Q cannot access or modify the page during the eviction process? How do you avoid a race between P evicting Q's frame and Q faulting the page back in?

所以我page_fault的时候，如果我打算evict这个frame了，这个frame一开始一直都是可以用的，数据都是存的是对的，直到我把它从pagedir中clear掉，当我clear掉之后，我已经完全把Q的这个数据存放在内存中，并且标记它是可用的了。这之后我才会让 P 有机会去访问这个内存（清空或别的问题）

如果这时候 Q 也page_fault了，可能的情况是：

1、没有让 Q 完全 evict掉，那么这时候page_fault的必然不是这个页，因为这个页是可以正常访问的。

2、Q已经evict掉这个页了，那么所有信息都是够的，我可以尝试再重新载入这个页，注意到alloc_frame是加锁的，所以P和Q的访问不会冲突。

> B7: Suppose a page fault in process P causes a page to be read from the file system or swap. How do you ensure that a second process Q cannot interfere by e.g. attempting to evict the frame while it is still being read in?

首先SWAP和FILE SYS在两个不同的分区，而磁盘访问是肯定得加锁的，这是pintos的block_read/write页有的。

对于 SWAP 分区的相关写/读也是加锁，让任何 SWAP 都得明确那些地方是可以写的，用一个bitmap来存下来可以写的磁盘的地方。

> B8: Explain how you handle access to paged-out pages that occur during system calls. Do you use page faults to bring in pages (as in user programs), or do you have a mechanism for "locking" frames into physical memory, or do you use some other design? How do you gracefully handle attempted accesses to invalid virtual addresses?

这地方实现得不够优雅哈，

我就是直接在syscall之前，进行了一波访问，看看会不会出现不合法的情况

有些情况是合法的但是可能不太好处理的，比如如果这只是个没有被加载的页（或许是文件，或许在SWAP里面），那我其实是合理的，合法的访问，那我在检查的时候去加载一下看看。

这样就可以没有 `invalied vatual address` 了

然后后面继续读的时候，改page_fault就page_fault。

RATIONALE

> B9: A single lock for the whole VM system would make synchronization easy, but limit parallelism. On the other hand, using many locks complicates synchronization and raises the possibility for deadlock but allows for high parallelism. Explain where your design falls along this continuum and why you chose to design it this way.

我给 `frame` 的处理加了锁，给 `swap` 的处理加了锁

这是合理的，因为除此之外，对于 supplied_page_table 来说，这是每个线程独有的，那么一个线程就可以自己去处理这些东西，这个并不是需要并发实现的。

对 `frame` ，因为是公用的，而且需要时钟算法去访问全局的信息，那么加锁是很合理的。

对于 `swap` ，磁盘的访问也需要直到访问哪些地方，这个得有互斥。