Project 3a: Virtual Memory

Preliminaries

> Fill in your name and email address.

Ivory E. Si 2000012957@stu.pku.edu.cn

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

The small capacity of memory system of pintos prohibits many fancy memory management data structures.(e.g. buddy system)

And I am weak in designing a synchronization architecture for OS to improve performance. Could TA give me some advices or materials to learn?

> 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.

Three data structure

Supplemental Page Table: Use the **pagedir** in thread structure. The **page table** itself used by **MMU** is a radix tree. It's a beautiful structure that reaches a good trade-off between complexity of request and modify.

Frame Table : A hash table (**frames** in vm/frame.c) whose entry is a frame page structure recording what virtual address in which thread is occupying this frame.

Swap Space Slot: A bitmap to record whether the sector is used (**slot_bitmap** in vm/swap.c)

ALGORITHMS

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

If a frame is allocated to a thread at some virtual address, use look_page() in **pagedir.c** to look up a **page table entry**(PTE).

If the page is swapped out of the memory to the disk, we set PTE_P (the last bit) to zero. And remaining 31 bits can be used to record information for disk sector number.

Some bits are also used to help implement mmap function.

When the PTE is used to save information on swap space. The PTE_P is set to zero first. And PAGE_LOAD is set to 1 to show that it is a valid entry for information on dist.

The 1(PTE_W) and 2(PTE_U) bits are saved for recovering state for original PTE.

If this page is saved in swap space, the PAGE_SWAP is set to 1. And if PAGE_FILE is set, it means that the page is saved on a file on the disk.

The sixth bit is original PTE_D(dirty bit). This bit is used for page replacement. We keep this bit zero to avoid some potential error.

The sector number or mapid(for lab3b) is saved in the range from bit 31 to bit 9.

31-9	8,7	6	5	4	3	2	1	0
SECTOR_NUMBER/MAPID	RESERVE	0	PAGE_FILE	PAGE_SWAP	PAGE_LOAD	PTE_U	PTE_W	PTE_P
depend	any	0	0/1	0/1	1	keep	keep	0

> 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?

Use the page table maintained by MMU. The code does not change the dirty bit. So it doesn't need synchronization.

In the replacement algorithm, the access bit is set to zero. If it is accessed again(set by MMU), it is not to be swapped out of memory.

SYNCHRONIZATION

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

Lock the frame table when doing frame allocation or replacement.

We lock the whole process of page table maintenance. So when another user process is finding page to be replaced by visiting current process's page table, it won't get wrong page table entry.

RATIONALE

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

The page table is a radix tree itself. It balances the time to request a page and space to save metadata for virtual address information.

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.

swap_slot in **swap.c**

```
static struct block *swap_block; /* block object in file system */
static struct bitmap *slot_bitmap; /* a bitmap to mark all the sector state */
static uint32_t sector_for_page;
/* how many sectors to store a page, actually the number is 4096 / 512 = 8 */
static struct semaphore slot_sema;
/* how many page can be stored in swap slot, the system will wait if no swap
memory to store the page */
static struct lock slot_lock;
/* swap lock to ensure exclusive access. swap slot is a global data structure */
```

frame table in **frame.c**

```
static struct hash frames; /* a hash table for all the physical frame */

struct lock frame_hash_lock; /* a lock to ensure exclusive access */

struct frame_page {
    struct hash_elem hash_elem; /* hash element for hash table*/
    uintptr_t addr; /* physical address */
    struct thread* t; /* thread which is using this frame*/
    void *vaddr; /* virtual address mapped to this frame */
};
```

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.

We use the access bit changed by CPU to implement clock algorithm.

Check the dirty bit first. Then check the access bit (it must be true if we have not modify it). If the PTE_A is false, it means that the page misses its second chance. We swap the current page to disk and return this page to the page-demanding process.

If the PTE_A is ture, we set the bit to false and give this frame a second chance.

> 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?

Change Q's page table entry for the virtual memory mapping to the frame.

The modify process are protected by **pgtbl_lock** of Q. PTE_P is changed to zero. The next time of access will cause a page fault.

After the modification in **swap_to_disk**. The page table entry record the start sector number in swap slot to recover the page in the future.

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.)

We keep the same sequence in acquiring locks anywhere (alloc_lock, frame_hash_lock, pgtbl_lock of each thread, slot_lock), which avoids circular wait.

One exception is in check_valid function. We find the PTE first. (Because this PTE is not mapped to any valid physical frame. It won't bother the execution of finding frame function.)

> 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?

swap_to_disk writes the content in Q's memory to swap block. Then turn **PTE_P** to zero to inform MMU. After that Q starts to have possibility to generate a page fault for this address.

If Q's some other virtual address generates a page fault, P's eviction won't be interfered.

> 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?

The frame replacement is protected by a single lock. Q finishes its recover for some virtual address before P starts to find a frame to recover its memory content.

> 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?

Use page fault mechanism to bring in valid address.

invalid virtual addresses are unrecoverable and rejected by **check_valid** function.

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.

Use lock for each page_table for each thread

Global data structures frame_table and slot_table must be protected by locks

And the page_table setting process is protected by alloc_lock.

If only a small amount memory is occupied, every process can run without raising page fault.

But if the whole system runs out of memory, some overhead on memory swapping in and out is unavoidable. The frame table is the critical section in replacement. Some parallelism applied may improve its performance. But the relationship behind dependencies is really complicated. It is better to keep simple design in such simple hardware(single core processor).