CS302 OS Week6 Assignment - Report

Name: 刘仁杰
SID: 11911808

1. explain what happens when the process accesses a memory page not present in the physical memory.

When the process accesses a memory page not present in the physical memory, it can be roughly divided into two cases.

- One is that when the process first access a page that has not been allocated in physical memory, then OS will allocate a new page in physical memory and may flush some pages out to disk according to page replacement algorithm if the physical memory is full.
- The other is that if a page is not present and has been swapped to disk, the OS will need to swap the page into memory in order to service the page fault. When the OS receives a page fault for a page in this case, it looks in the PTE to find the address, and issues the request to disk to fetch the page into memory. If the memory is full, OS will need to swap some pages out according to page replacement algorithm.

2. Consider the following reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1. Consider 4 pages. What are the number of page faults with the following policy: Optimal (MIN), LRU, FIFO.

- MIN: #page faults=8

page	**7**	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
0	7					3				h		h		1			h			h
1		0			h		h				h					h			h	
2			1					4												
3				2					h				h		h			7		

- LRU: #page faults=8

page	**7**	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
0	7					3				h		h						7		
1		0			h		h				h					h			h	
2			1					4						1			h			h
3				2					h				h		h					

- FIFO: #page faults=10

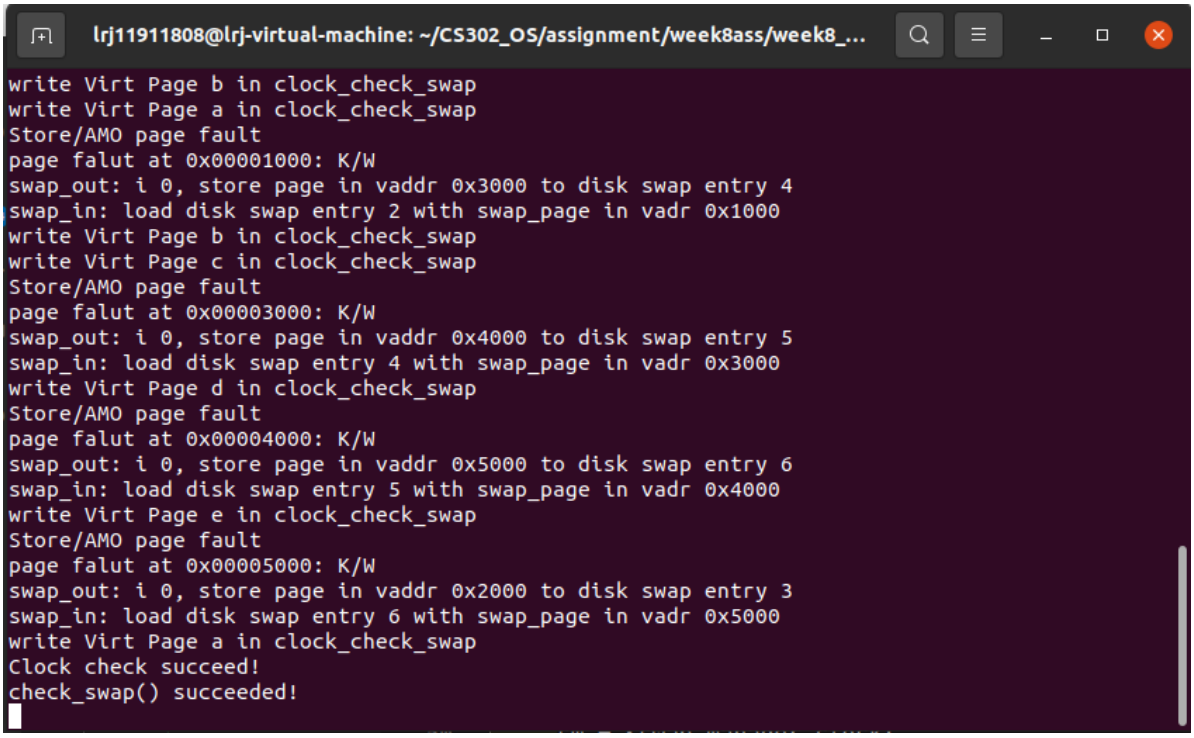
page	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
0	7					3				h		h			2					
1		0			h		h	4										7		
2			1								0					h			h	
3				2					h				h	1			h			h

3. Realize Clock algorithm in swap_clock.c

```
static int
_clock_init_mm(struct mm_struct *mm)
{
    // init page list, set sm_priv and current pointer
    list_init(&pra_list_head);
    mm->sm_priv = &pra_list_head;
    curr_ptr = &pra_list_head;
    return 0;
}

static int
_clock_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
{
    // insert the incoming page to the next position of current pointer
    list_entry_t *entry = &(page->pra_page_link);
    assert(entry != NULL && curr_ptr != NULL);
    list_add(curr_ptr, entry);
    return 0;
}
```

```
_clock_swap_out_victim(struct mm_struct *mm, struct Page **ptr_page, int in_tick)
{
    list_entry_t *head = (list_entry_t*) mm->sm_priv;
    assert(head != NULL && in_tick == 0);
    // if the page list is empty, just do nothing and return
    if (list_next(head) == head)
    {
        *ptr_page = NULL;
        return 0;
    }
    curr_ptr = list_next(curr_ptr);
    bool accessed;
    do
    {   // move the current pointer to the previous one
        curr_ptr = list_prev(curr_ptr);
        // if current pointer is head, just skip it
        if (curr_ptr == head)
        {
            continue;
        }
        *ptr_page = le2page(curr_ptr, pra_page_link);
        pte_t *pte = get_pte(mm->pgdir, (*ptr_page)->pra_vaddr, 0);
        accessed = *pte & PTE_A;
        // is accessed, skip it and set the reference to unaccessed
        if (accessed)
        {
            *pte &= ~PTE_A;
        }
    } while (accessed);
    // get the target, delete it from the list and return
    curr_ptr = list_prev(curr_ptr);
    list_del(list_next(curr_ptr));
    return 0;
}
```



4. Realize LRU algorithm in swap_lru.c

```c
static int
_lru_init_mm(struct mm_struct *mm)
{
    // init page list, set sm_priv
    list_init(&pra_list_head);
    mm->sm_priv = &pra_list_head;
    return 0;
}

static int
_lru_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
{
    // insert the incoming page to the next position of head pointer
    list_entry_t *head = (list_entry_t *)mm->sm_priv;
    list_entry_t *entry = &(page->pra_page_link);
    assert(entry != NULL && head != NULL);
    list_add(head, entry);
    return 0;
}
```

```c
static int
_lru_swap_out_victim(struct mm_struct *mm, struct Page **ptr_page, int in_tick)
{
    list_entry_t *head = (list_entry_t*) mm->sm_priv;
    assert(head != NULL && in_tick == 0);
    // if the page list is empty, just do nothing and return
    if (list_prev(head) == head)
    {
        *ptr_page = NULL;
        return 0;
    }
    curr_ptr = list_prev(head);
    list_entry_t *target = NULL;
    int times = __INT_MAX__;
    do
    {
        *ptr_page = le2page(curr_ptr, pra_page_link);
        // get the constant in page
        int temp = *(unsigned char *)(*ptr_page)->pra_vaddr;
        // if the used times is smaller than the current one,
        // it will be the new target to swap out
        if (temp < times)
        {
            times = temp;
            target = curr_ptr;
        }
        // move the current pointer to the previous one
        curr_ptr = list_prev(curr_ptr);
    } while (curr_ptr != head);
    // get the target, delete it from the list and return
    *ptr_page = le2page(target, pra_page_link);
    list_del(target);
    return 0;
}
```

```
lrj11911808@lrj-virtual-machine: ~/CS302_OS/assignment/week8ass/week8_...
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
write Virt Page 1 in lru_check_swap
Store/AMO page fault
page falut at 0x00001000: K/W
swap_out: i 0, store page in vaddr 0x4000 to disk swap entry 5
swap_in: load disk swap entry 2 with swap_page in vadr 0x1000
write Virt Page 4 in lru_check_swap
Store/AMO page fault
page falut at 0x00004000: K/W
swap_out: i 0, store page in vaddr 0x2000 to disk swap entry 3
swap_in: load disk swap entry 5 with swap_page in vadr 0x4000
write Virt Page 4 in lru_check_swap
write Virt Page 5 in lru_check_swap
write Virt Page 2 in lru_check_swap
Store/AMO page fault
page falut at 0x00002000: K/W
swap_out: i 0, store page in vaddr 0x3000 to disk swap entry 4
swap_in: load disk swap entry 3 with swap_page in vadr 0x2000
write Virt Page 3 in lru_check_swap
Store/AMO page fault
page falut at 0x00003000: K/W
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
LRU check succeed!
check_swap() succeeded!
```