Assignment 9

1. [50pts] Read Chapter 21 of “Three Easy Pieces” (https://pages.cs.wisc.edu/~remzi/OSTEP/vm-beyondphys.pdf) and explain what happens when the process accesses a memory page not present in the physical memory.

A: First, the OS will check TLB and find it not here. Then, it will access the page table in the memory and find the PTE. The present bit of PTE is 0 which means it's not in physical memory. Thus, page fault happens. Then OS needs to allocate a physical page frame to map the VPN in the PTE and load it into memory. If memory is already full, then a page replacement algorithm is called to swap out and swap in the page. Finally, the memory page accessing will try again.

2.Realize Clock algorithm in swap_clock.c

Code:

```
13  static int
14  _clock_init_mm(struct mm_struct *mm)
15  {
16      list_init(&pra_list_head);
17      mm->sm_priv = &pra_list_head;
18      curr_ptr = &pra_list_head;// curr_ptr record the current pointer, initialized as head
19      return 0;
20  }
21
22  static int
23  _clock_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
24  {
25      list_entry_t *head=(list_entry_t*) mm->sm_priv;
26      list_entry_t *entry=&(page->pra_page_link);
27
28      list_add(curr_ptr, entry);
29
30      struct Page *ptr_page = le2page(entry,pra_page_link);
31      pte_t* ptep = get_pte(mm->pgdir, ptr_page->pra_vaddr, 0);
32      *ptep &= ~PTE_A;// set Access of this page as 0
33      return 0;
34  }
```

```c
static int
_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);
    assert(in_tick == 0);
    if(curr_ptr == head){
        curr_ptr = list_prev(curr_ptr);
        if(curr_ptr == head){
            *ptr_page = NULL;
            return 0;
        }
    }// judge the list of swappable pages is not empty
    while(1){
        if(curr_ptr== head) curr_ptr = list_prev(curr_ptr); // special judge to avoid head
        list_entry_t *cur = curr_ptr;
        struct Page *page = le2page(cur, pra_page_link);
        pte_t *ptep = get_pte(mm->pgdir, page->pra_vaddr, 0);
        if (!(*ptep & PTE_A)) { // Access = 0
            *ptr_page = page;
            list_del(cur);
            curr_ptr = list_prev(curr_ptr);
            return 0;
        }else{
            *ptep &= ~PTE_A; // set Access of this page as 0
        }
        curr_ptr = list_prev(curr_ptr);// current pointer move to next page
        }
    return 0;
```

Result:

```
write Virt Page a in clock_check_swap
write Virt Page d in clock_check_swap
write Virt Page b in clock_check_swap
write Virt Page e in clock_check_swap
Store/AMO page fault
page falut at 0x00005000: K/W
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2
write Virt Page b in clock_check_swap
write Virt Page a in clock_check_swap
Store/AMO page fault
page falut at 0x00001000: K/W
swap_out: i 0, store page in vaddr 0x3000 to disk swap entry 4
swap_in: load disk swap entry 2 with swap_page in vadr 0x1000
write Virt Page b in clock_check_swap
write Virt Page c in clock_check_swap
Store/AMO page fault
page falut at 0x00003000: K/W
swap_out: i 0, store page in vaddr 0x4000 to disk swap entry 5
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
write Virt Page d in clock_check_swap
Store/AMO page fault
page falut at 0x00004000: K/W
swap_out: i 0, store page in vaddr 0x5000 to disk swap entry 6
swap_in: load disk swap entry 5 with swap_page in vadr 0x4000
write Virt Page e in clock_check_swap
Store/AMO page fault
page falut at 0x00005000: K/W
swap_out: i 0, store page in vaddr 0x2000 to disk swap entry 3
swap_in: load disk swap entry 6 with swap_page in vadr 0x5000
write Virt Page a in clock_check_swap
Clock check succeed!
check_swap() succeeded!
QEMU: Terminated
xjn12110714@xjn12110714-virtual-machine:~/Desktop/Assignment9$
```