1. When a process accesses a memory page not present in the physical memory, a page fault will happen, and a page-fault handler will start running. Firstly, the OS will find a physical frame for the soon-to-be-faulted-in page to reside within. If there is no such physical frames, the swap algorithm runs and kick some pages from the memory to free some space. With a physical frame in hand, the handler then issues the I/O request to read in the page from swap space. Finally, when that slow operation completes, the OS updates the page table and retries the instruction. The retry will result in a TLB miss, and then, upon another retry, a TLB hit, at which point the hardware will be able to access the desired item.
2. Optimal:8 LRU:8 FIFO:10
3.

swap_clock.c

```c
static int
_clock_init_mm(struct mm_struct *mm)
{
    //TODO
    list_init(&pra_list_head);
    mm->sm_priv = &pra_list_head;

    //make current pointer point to the head
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    curr_ptr = head;

    return 0;
}
```

```c
static int
_clock_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
{
    //TODO
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    list_entry_t *entry=&(page->pra_page_link);

    assert(entry != NULL && head != NULL);

    //link the arrival page in front of the current pointer
    list_add(curr_ptr -> prev, entry);

    return 0;
}
```

```c
static int
_clock_swap_out_victim(struct mm_struct *mm, struct Page ** ptr_page, int in_tick)
{
    //TODO
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    assert(head != NULL);
    assert(in_tick==0);
    while (1) {
        //When the current pointer points to the head, move on
        if (curr_ptr == head) {
            curr_ptr = list_next(curr_ptr);
        }
        //Get the visited bit of the pointed page
        struct Page *ptr = le2page(curr_ptr, pra_page_link);
        pte_t *ptep = get_pte(mm->pgdir, ptr->pra_vaddr, 0);
        //move the current pointer to the next
        curr_ptr = list_next(curr_ptr);
        if (!(*ptep & PTE_A)) {//If not visited (visited bit is 0)
            //Unlink the earliest arrival page in front of pra_list_head queue
            //set the addr of this page to ptr_page
            list_entry_t *le = list_prev(curr_ptr);
            if (le != head) {
                list_del(le);
                *ptr_page = ptr;
            } else {
                *ptr_page = NULL;
            }
            break;
        }
        else {//If visited (visited bit is 1)
            *ptep &= ~PTE_A; //Set the visited bit to 0
        }
    }
    return 0;
}
```

```
ljj11912021@ljj11912021-virtual-machine: ~/Desktop/week8_exe/week8_exe
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!
```

4.

```c
static int
_lru_init_mm(struct mm_struct *mm)
{
    //TODO
    list_init(&pra_list_head);
    mm->sm_priv = &pra_list_head;

    //let current pointer point to the head
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    curr_ptr = head;

    return 0;
}

static int
_lru_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
{
    //TODO
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    list_entry_t *entry=&(page->pra_page_link);

    assert(entry != NULL && head != NULL);

    //link the most recent arrival page at the back of the pra_list_head queue.
    list_add(head, entry);

    return 0;
}
```

```c
static int
_lru_swap_out_victim(struct mm_struct *mm, struct Page ** ptr_page, int in_tick)
{
    //TODO
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    assert(head != NULL);
    assert(in_tick==0);

    //let pointer le point to the next of head
    list_entry_t *le = head;
    le = list_next(le);
    //calculate the first page's constant as the minimal one
    struct Page *start_page = le2page(le, pra_page_link);
    int min = *(unsigned char *)start_page->pra_vaddr;
    //traverse the list
    while(le != head) {
        //calculate the current page's constant as temp
        struct Page *curr_page = le2page(le, pra_page_link);
        int temp = *(unsigned char *)curr_page->pra_vaddr;
        //pick the smaller constant as min and make curr_ptr
        //make curr_ptr point to the corresponding page
        if (temp < min) {
            min = temp;
            curr_ptr = le;
        }
        //move the pointer le to the next
        le = list_next(le);
    }
    //unlink the page pointed by curr_ptr
    //which is the one with the smallest constant
    list_entry_t *result = curr_ptr;
    if (result != head) {
        list_del(result);
        *ptr_page = le2page(result, pra_page_link);
    } else {
        *ptr_page = NULL;
    }
    return 0;
}
```

```
ljj11912021@ljj11912021-virtual-machine: ~/Desktop/week8_...
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!
```