# Week 8 Assignment

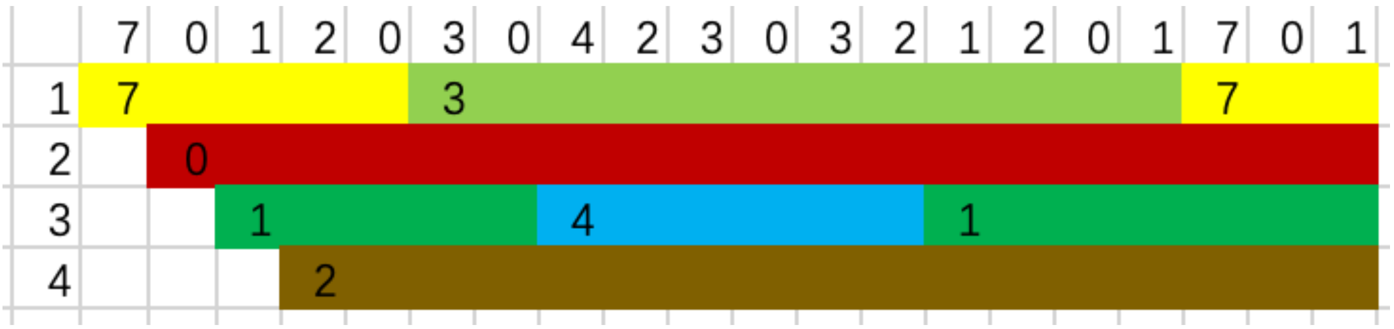
## 1

First of all, the operating system will find a physical frame for the page to be swapped in.  After the physical frame is obtained, the processing sequence issues an I/O request to read the page from the swap space. Finally, when the slow operation is complete, the operating system updates the page table and retries the instructions. Retry will cause the TLB to miss, and then whenretry, the TLB will hit, and the hardware will be able to access the desired value。

## 2

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

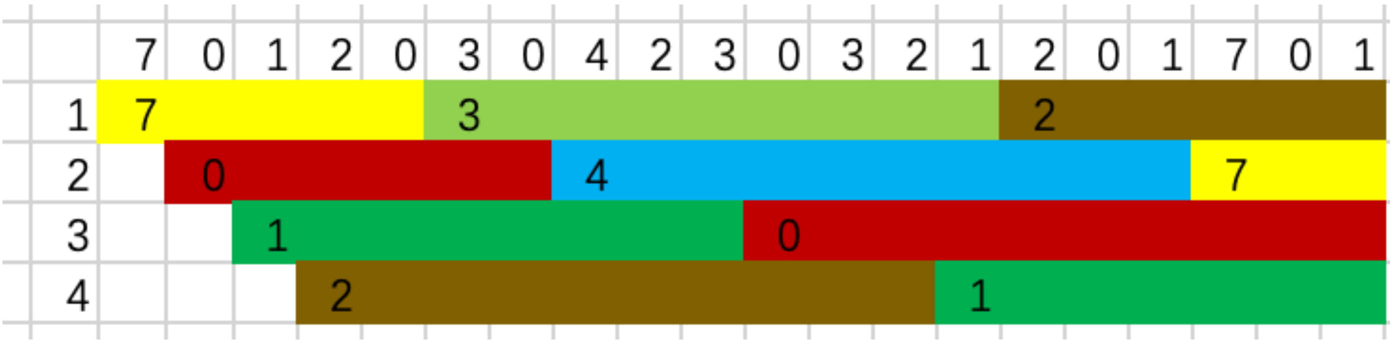
### Optimal



| | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7 | | | | | 3 | | | | | | | | | | | | 7 | | |
| 2 | | 0 | | | | | | | | | | | | | | | | | | |
| 3 | | | 1 | | | | | 4 | | | | | | 1 | | | | | | |
| 4 | | | | 2 | | | | | | | | | | | | | | | | |

8

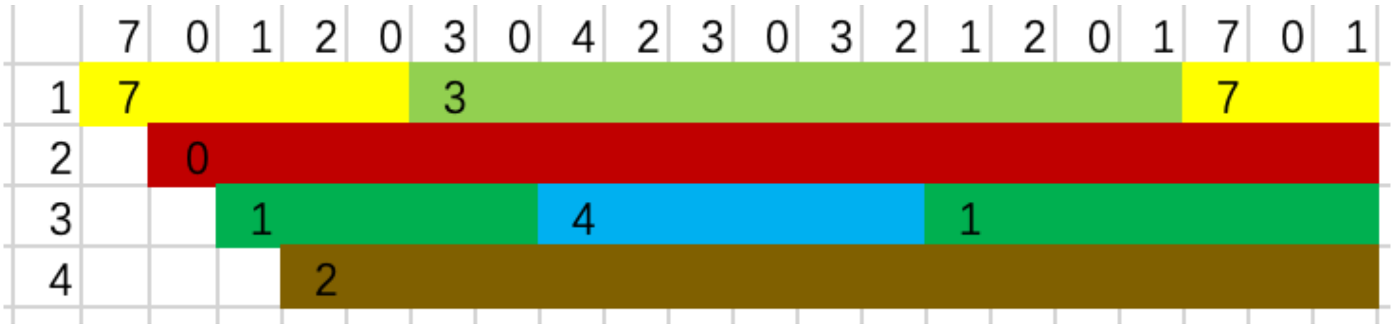
### FIFO



| | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7 | | | | | 3 | | | | | | | | | 2 | | | | | |
| 2 | | 0 | | | | | | 4 | | | | | | | | | | 7 | | |
| 3 | | | 1 | | | | | | | | 0 | | | | | | | | | |
| 4 | | | | 2 | | | | | | | | | | 1 | | | | | | |

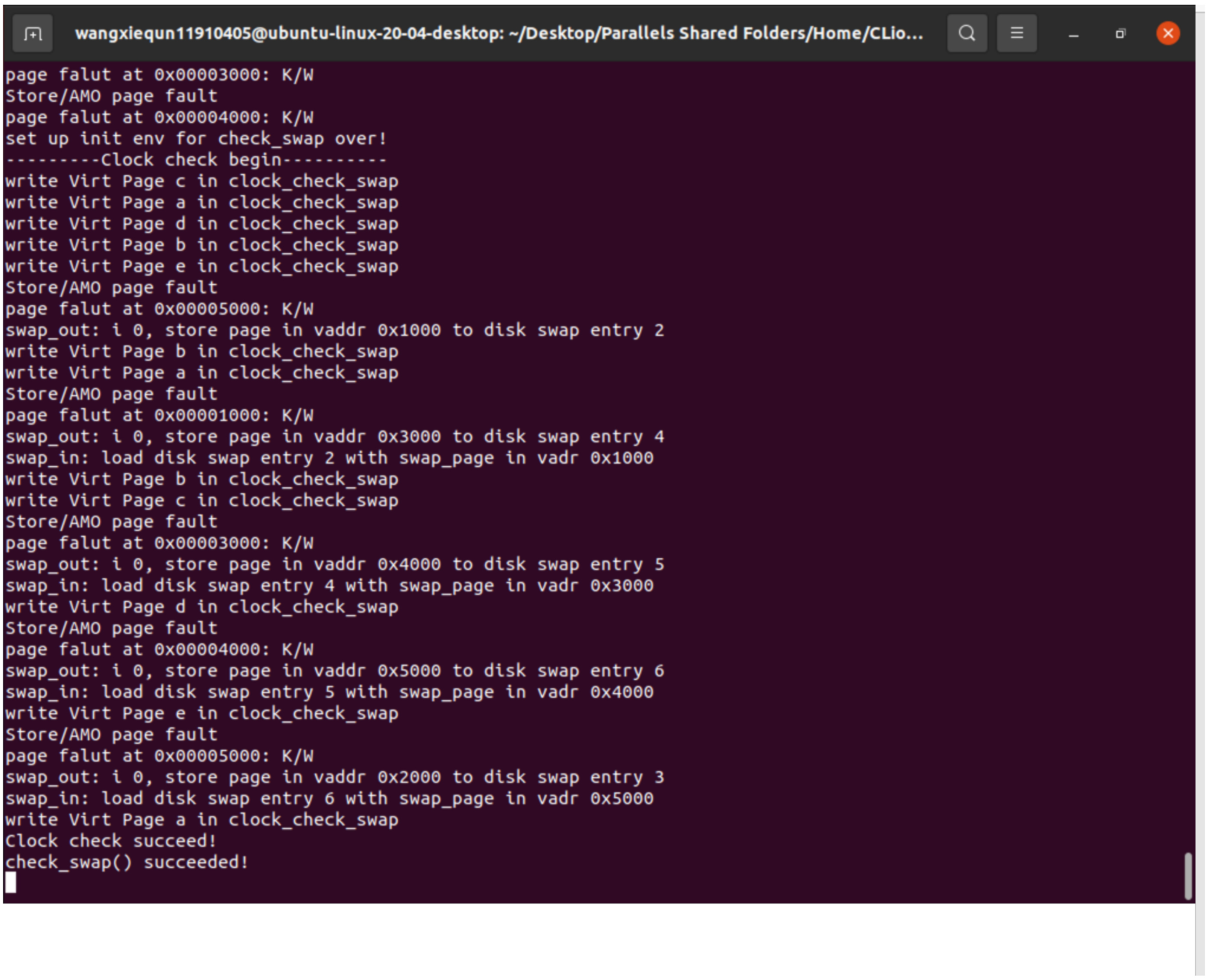
10

### LRU

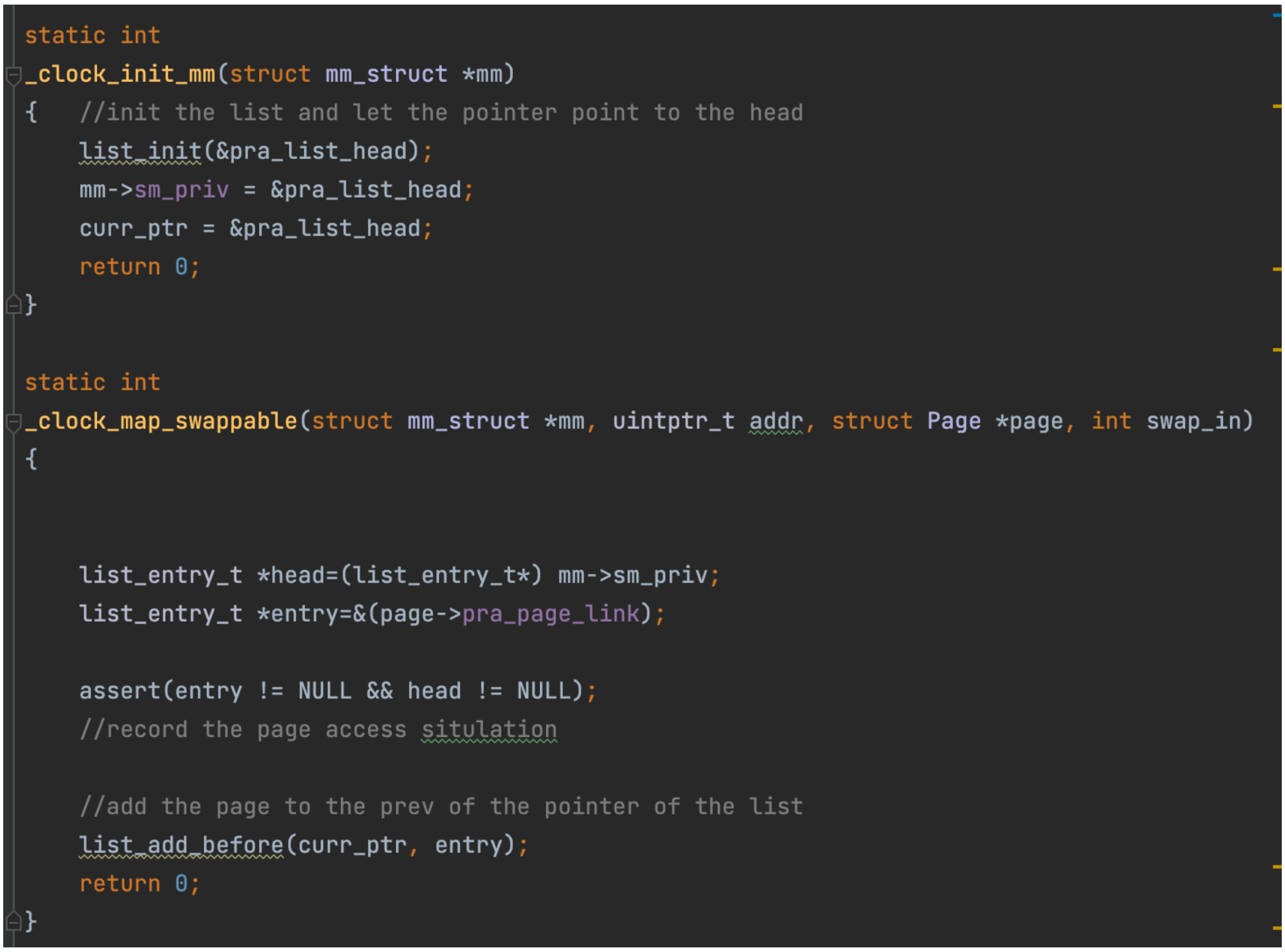


8

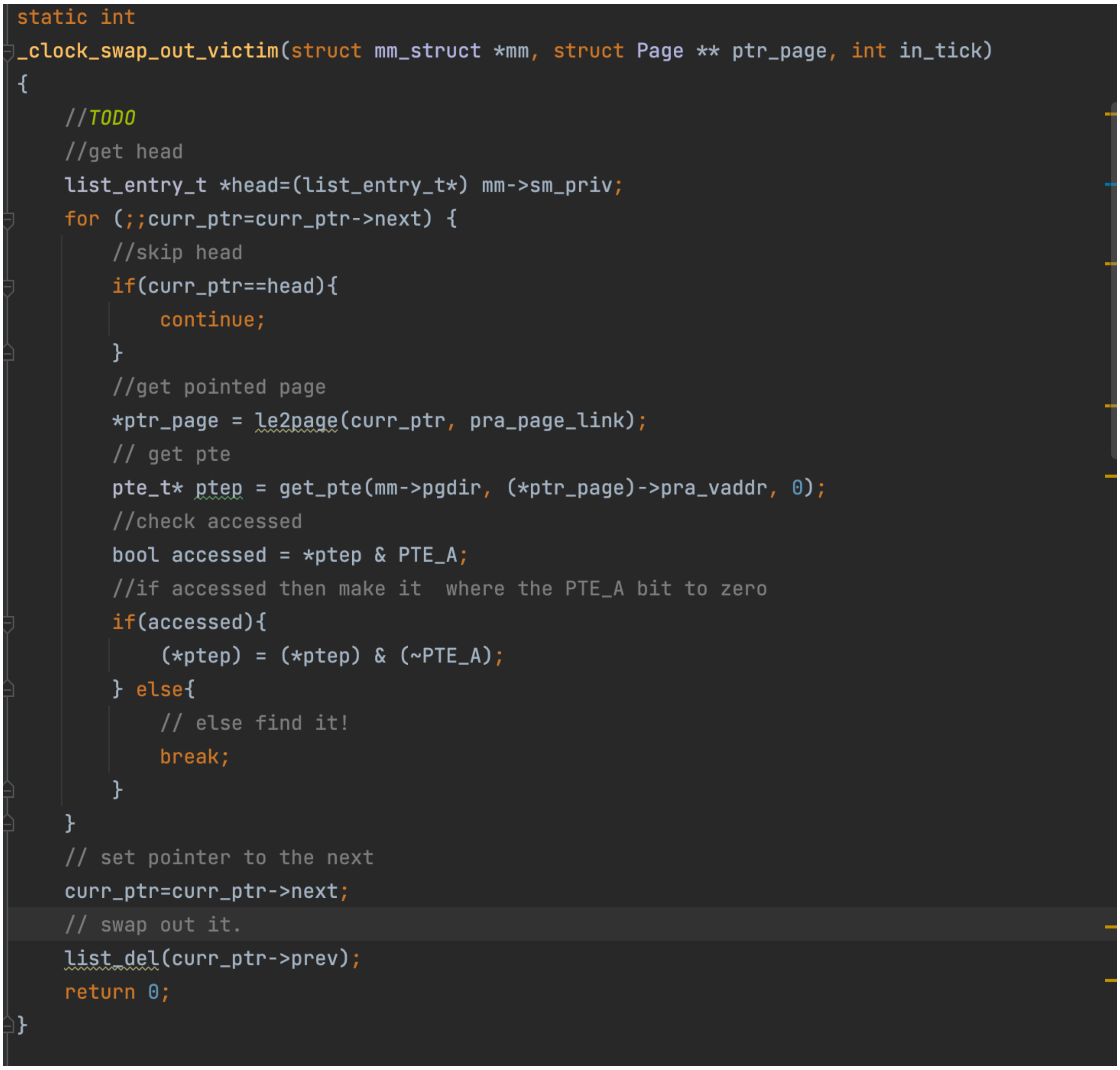
## 3



```
page falut at 0x00003000: K/W
Store/AMO page fault
page falut at 0x00004000: K/W
set up init env for check_swap over!
---------Clock check begin----------
write Virt Page c in clock_check_swap
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!
```

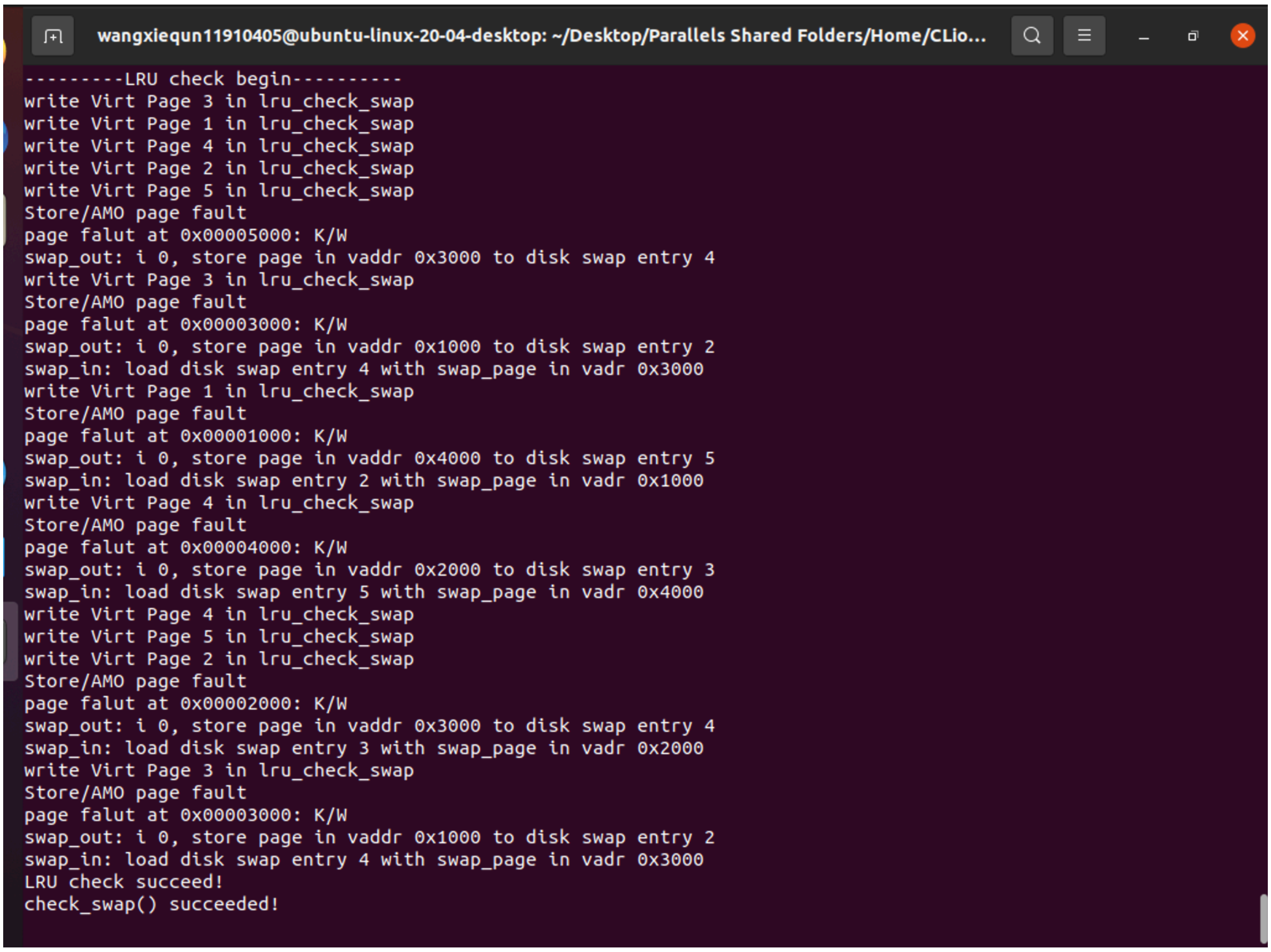


```c
static int
_clock_init_mm(struct mm_struct *mm)
{   //init the list and let the pointer point to the head
    list_init(&pra_list_head);
    mm->sm_priv = &pra_list_head;
    curr_ptr = &pra_list_head;
    return 0;
}

static int
_clock_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
{



    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    list_entry_t *entry=&(page->pra_page_link);

    assert(entry != NULL && head != NULL);
    //record the page access situlation

    //add the page to the prev of the pointer of the list
    list_add_before(curr_ptr, entry);
    return 0;
}
```

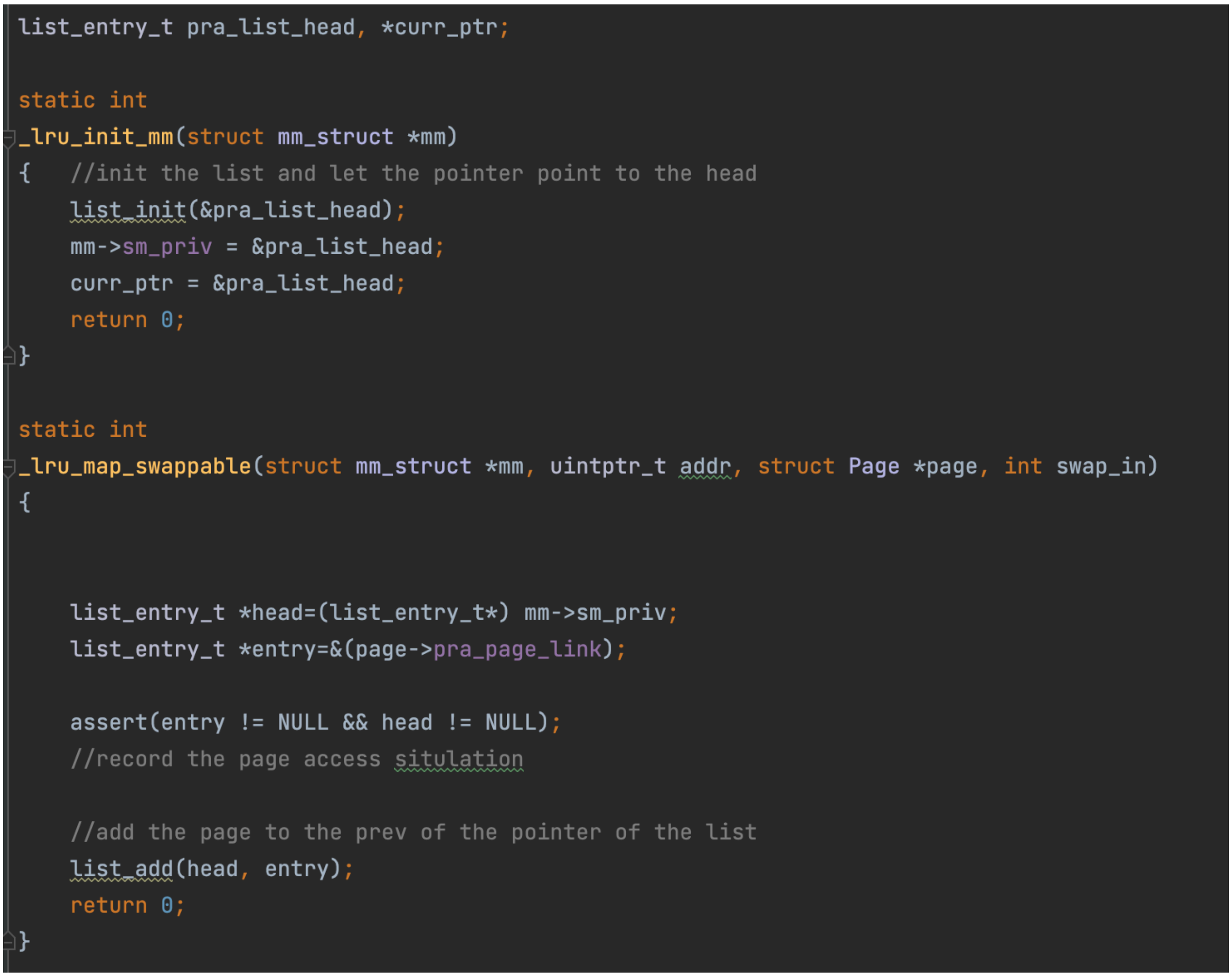


```c
static int
_clock_swap_out_victim(struct mm_struct *mm, struct Page ** ptr_page, int in_tick)
{
    //TODO
    //get head
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    for (;;curr_ptr=curr_ptr->next) {
        //skip head
        if(curr_ptr==head){
            continue;
        }
        //get pointed page
        *ptr_page = le2page(curr_ptr, pra_page_link);
        // get pte
        pte_t* ptep = get_pte(mm->pgdir, (*ptr_page)->pra_vaddr, 0);
        //check accessed
        bool accessed = *ptep & PTE_A;
        //if accessed then make it  where the PTE_A bit to zero
        if(accessed){
            (*ptep) = (*ptep) & (~PTE_A);
        } else{
            // else find it!
            break;
        }
    }
    // set pointer to the next
    curr_ptr=curr_ptr->next;
    // swap out it.
    list_del(curr_ptr->prev);
    return 0;
}
```

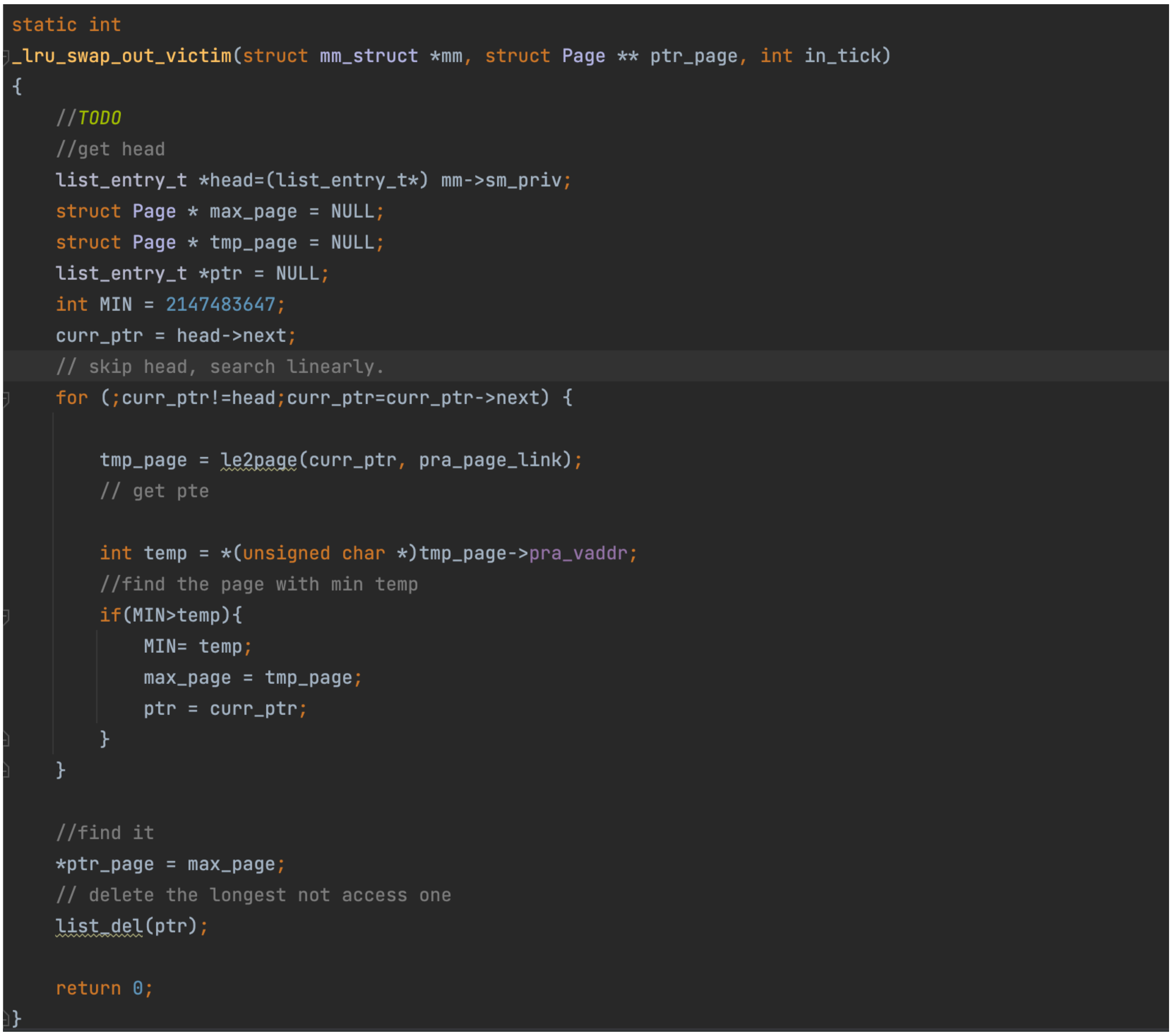
## 4


wangxiequn11910405@ubuntu-linux-20-04-desktop: ~/Desktop/Parallels Shared Folders/Home/CLio...

```
---------LRU check begin----------
write Virt Page 3 in lru_check_swap
write Virt Page 1 in lru_check_swap
write Virt Page 4 in lru_check_swap
write Virt Page 2 in lru_check_swap
write Virt Page 5 in lru_check_swap
Store/AMO page fault
page falut at 0x00005000: K/W
swap_out: i 0, store page in vaddr 0x3000 to disk swap entry 4
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
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!
```



```c
list_entry_t pra_list_head, *curr_ptr;

static int
_lru_init_mm(struct mm_struct *mm)
{   //init the list and let the pointer point to the head
    list_init(&pra_list_head);
    mm->sm_priv = &pra_list_head;
    curr_ptr = &pra_list_head;
    return 0;
}

static int
_lru_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
{



    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    list_entry_t *entry=&(page->pra_page_link);

    assert(entry != NULL && head != NULL);
    //record the page access situlation

    //add the page to the prev of the pointer of the list
    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
    //get head
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
    struct Page * max_page = NULL;
    struct Page * tmp_page = NULL;
    list_entry_t *ptr = NULL;
    int MIN = 2147483647;
    curr_ptr = head->next;
    // skip head, search linearly.
    for (;curr_ptr!=head;curr_ptr=curr_ptr->next) {

        tmp_page = le2page(curr_ptr, pra_page_link);
        // get pte

        int temp = *(unsigned char *)tmp_page->pra_vaddr;
        //find the page with min temp
        if(MIN>temp){
            MIN= temp;
            max_page = tmp_page;
            ptr = curr_ptr;
        }
    }

    //find it
    *ptr_page = max_page;
    // delete the longest not access one
    list_del(ptr);

    return 0;
}
```