Assignment 6

12110517 Zhongzhiyuan

Q1

- The hardware checks whether the page is in physical memory by looking up the page table. If the present bit of the PTE is 0, the page is not in the memory but in the disk(or swap space). The act of accessing a page that is not in physical memory is commonly referred to as a page fault. Upon a page fault, the OS is invoked to service the page fault. A particular piece of code, known as a page-fault handler, runs and must service the page fault.
- If a page has been swapped to disk, the OS will need to swap the page into memory in order to service the page fault. The OS looks in the PTE to find the disk address, and issues the request to disk to fetch the page into memory.
- When the disk I/O completes, the OS will then update the page table to mark the page as present, update the PFN field of the page-table entry (PTE) to record the in-memory location of the newly-fetched page, and retry the instruction. This next attempt may generate a TLB miss, which would then be serviced and update the TLB with the translation (one could alternately update the TLB when servicing the page fault to avoid this step). Finally, a last restart would find the translation in the TLB and thus proceed to fetch the desired data or instruction from memory at the translated physical address
- Note that while the I/O is in flight, the process will be in the blocked state. Thus, the OS will be free to run other ready processes while the page fault is being serviced. Because I/O is expensive, this overlap of the I/O (page fault) of one process and the execution of another is yet another way a multiprogrammed system can make the most effective use of its hardware.
- If the memory is full, the OS might like to first page out one or more pages to make room for the new page(s) the OS is about to bring in. The process of picking a page to kick out, or replace is known as the page-replacement policy (LRU, FIFO, etc.).

Q2

three main functions in `swap_clock.c`:

- clock init and clock map swappable:

```
10  list_entry_t pra_list_head, *curr_ptr;
11
12  static int
13  _clock_init_mm(struct mm_struct *mm)
14  {
15      // TODO
16      list_init(&pra_list_head);
17      mm->sm_priv = &pra_list_head;
18      // initially curr ptr points to the head
19      curr_ptr = &pra_list_head;
20      return 0;
21  }
22
23  static int
24  _clock_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
25  {
26      // TODO
27      //  insert the new page to the previous entry of the curr_ptr
28      list_entry_t *entry = &(page->pra_page_link);
29      assert(entry != NULL && curr_ptr != NULL);
30      // list_del(curr_ptr->prev): the previous entry was deleted in _clock_swap_out_victim
31      list_add_before(curr_ptr, entry);
32      pte_t *ptep = get_pte(mm->pgdir, (page)->pra_vaddr, 0);
33      (*ptep) |= (PTE_A); // set access bit to 1
34      return 0;
35  }
```

- clock swap out victim

```
37  static int
38  _clock_swap_out_victim(struct mm_struct *mm, struct Page **ptr_page, int in_tick)
39  {
40      // TODO
41      list_entry_t *head = (list_entry_t *)mm->sm_priv;
42      assert(head != NULL && in_tick == 0);
43      while (true)
44      {
45          if (curr_ptr != head)
46          {
47              *ptr_page = le2page(curr_ptr, pra_page_link);
48              pte_t *ptep = get_pte(mm->pgdir, (*ptr_page)->pra_vaddr, 0);
49              if ((*ptep) & PTE_A)     // visit bit = 1
50                  (*ptep) &= (~PTE_A); // set to 0
51
52              else // visit bit = 0
53              {
54                  curr_ptr = curr_ptr->next;
55                  list_del(curr_ptr->prev); // see you again~ victim
56                  break;
57              }
58          }
59          curr_ptr = curr_ptr->next;
60      }
61      return 0;
62  }
```

- result

```
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
```