Lab 3

Rongxin Xia

1 Implementation of GCA

The idea is to check access and dirty digits to determine whether evict the frame

Algorithm 1 Global Value: Last (id recording last frame evicted)

```
Now = (Last + 1) % NFRAMES
for (i = 0; i<4×NFRAMES; i++) (at most go over 3 times the whole frame table ) do
  if The frame with id Now is allocated for the Page then
    if Access and Dirty digits of the corresponding page are both 0 then
      Last = Now
      return the frame with id Now
    else if Access digit is 1 and Dirty digit of the corresponding page is 0 then
      Change Access digit to 0
    else
      Change Dirty digit to 0
      Change the frame's dirty attribute to 1 (to find whether write back)
    end if
  end if
  Now = (Now + 1) % NFRAMES
end for
return SYSERR
```

Next page is the code for GCA.

```
int32 get_free_frame_gca(void){
  intmask mask;
  int32 curr_frameid = (last_frameid+1)%NFRAMES;
  frame_t *curr_frame;
  uint32 vpn;
  uint32 vd;
  vd_t *fault_vd;
  pd_t *curr_pd;
  pt_t *curr_pt;
  int32 i;

  mask=disable();
  // at most go over 3 times
  for(i=0; i<4*NFRAMES; i++){
    curr_frame = &frame_tab[curr_frameid];
    //here would be not free frame
    // if the frame is for page
    if(curr_frame->type == FRAME_PG){
      vpn = inverted_page_tab[curr_frameid].vpn;
      vd = VPN_TO_VD(vpn);
      fault_vd = (vd_t *)(&vd);
      curr_pd = proctab[currpid].prpdptr;
      curr_pt = (pt_t*)VPN_TO_VD(curr_pd[fault_vd->pd_offset].pd_base);
      // (0,0) case
      if((curr_pt[fault_vd->pt_offset].pt_acc == 0) &&
      (curr_pt[fault_vd->pt_offset].pt_dirty == 0)){
        curr_frame->type = -1; // change type of the frame
        last_frameid = curr_frameid;
        rm_frame_fifo(curr_frameid); //remove from FIFO queue
        restore(mask);
        return curr_frameid;
      //      10      case
      }else if((curr_pt[fault_vd->pt_offset].pt_acc == 1) &&
      (curr_pt[fault_vd->pt_offset].pt_dirty == 0)){
        curr_pt[fault_vd->pt_offset].pt_acc = 0;
      //      11      case
      }else if((curr_pt[fault_vd->pt_offset].pt_acc == 1) &&
      (curr_pt[fault_vd->pt_offset].pt_dirty == 1)){
        curr_pt[fault_vd->pt_offset].pt_dirty =0;
        curr_frame->dirty = 1;
      }
    }

    curr_frameid = (curr_frameid+1)%NFRAMES;

  }
  restore(mask);
  return SYSERR;
}
```

2 Performance

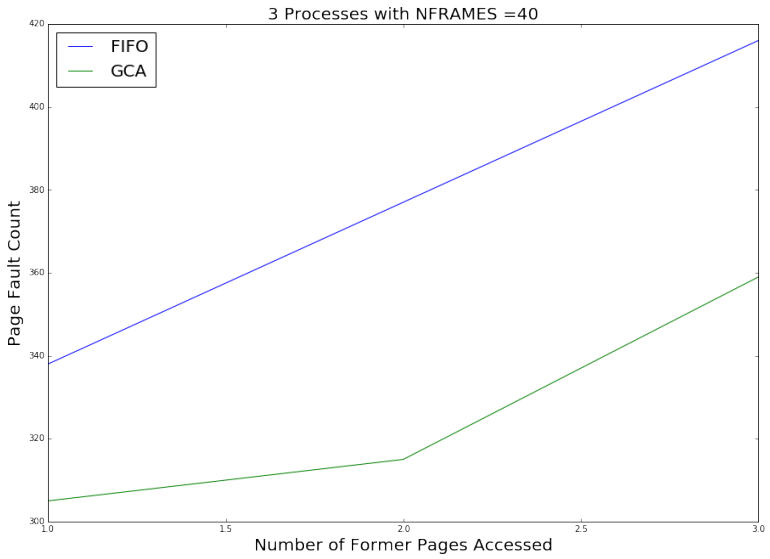
I test with the code modified from the test case provided to us. I make each loop try to access a former page and also a new page.

The code is

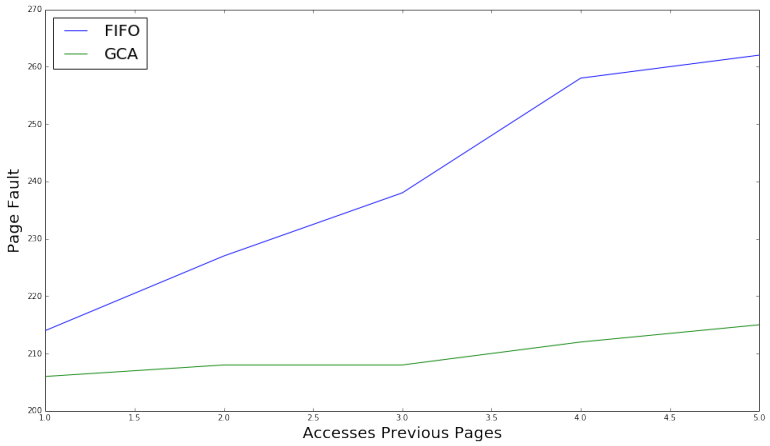
```
1
2  static void do_policy_test(void) {
3    uint32 npages = PAGE_ALLOCATION - 1;
4    uint32 nbytes = npages * PAGESIZE;
5    uint32 number =3;
6
7    kprintf("Running Page Replacement Policy Test, with NFRAMES = %d\n", NFRAMES);
8
9    char *mem = vgetmem(nbytes);
10   if (mem == (char*) SYSERR) {
11     panic("Page Replacement Policy Test failed\n");
12     return;
13   }
14
15   // Write data
16   for (uint32 i = 0; i<npages; i++) {
17     uint32 *p = (uint32*)(mem + (i * PAGESIZE));
18     // kprintf("Write Iteration [%3d] at 0x%08x\n", i, p);
19     //access a new page
20     uint32 v = get_test_value(p);
21     *p = v;
22     //access a old one
23     int j = i % number;
24     p = (uint32*)(mem+(j * PAGESIZE));
25     v = get_test_value(p);
26     *p = v;
27     sleepms(20); // to make it slower
28   }
29
30   if (vfreemem(mem, nbytes) == SYSERR) {
31     panic("Policy Test: vfreemem() failed.\n");
32   } else {
33 #if PAGE_REPLACEMENT == 1
34     kprintf("\nPage Replacement Policy Test Finished.\n");
35 #else
36     kprintf("\nPage Fault Handling Test Finished\n");
37 #endif
38     kprintf("Here NFRAMES = %d\n", NFRAMES);
39   }
40 }
```

Where number is the number of former pages accessed.

With 3 processes and **NFRAMES**=40, the number of former pages accessed are 1, 2, 3, 4, 5 (variable number in the code.) The page faults for FIFO is 333, 370, 396, 440, 570 and GCA is 305, 313, 341, 363, 424.



With 2 processes and **NFRAMES**=40, the number of former pages accessed are 1, 2, 3, 4, 5 (variable number in the code.) The page faults for FIFO is 214, 227, 238, 258, 262 and GCA is 206, 208, 208, 212, 215.



It shows GCA performs better than FIFO. It is because GCA tends to keep pages accessed often in the memory while FIFO may evict these pages because these pages are in the memory for a long time.