CS50300: Operating System

Lab 3

23 April, 2019

Name: Siddhartha Shankar Das
Email: das90@purdue.edu

1 GCA Implementation

The GCA algorithm sweeps through the resident pages while updating access bit and dirty bit of the page table entry. In addition, I have another global frame data structure which tracks dirty bit property of a frame. It is used to swap frame marked as dirty in back-store later on. Following code shows my overall C-Style pseudo code of my implementation of GCA.

```c
int32 get_frame_gca(void){
  int32 frameNo=(lframeNo+1)%NFRAMES;//next frame after previously selected last frame
  for(int i=0;i<SWEEP_TIMES*NFRAMES;i++){
    frame_entry=&frame_tab[frameNo];
    //only replace resident pages
    if(frame_entry->type==FRAME_PR){
      ptptr=get_page_table_entry(frame_entry);
      if(ptptr->pt_acc==0 && ptptr->pt_dirty==0){//0,0
        //remove this one
        lframeNo=frameNo;
        removeFromFrameList(frameNo);
        ptptr->pt_pres=0;//mark this frame as not present
        return frameNo;
      }
      else if(ptptr->pt_acc==1 && ptptr->pt_dirty==0){// 1,0
        ptptr->pt_acc=0;
      }
      else if(ptptr->pt_acc==1 && ptptr->pt_dirty==1){// 1,1
        ptptr->pt_dirty=0;
        frame_entry->dirty=1; //will swap this back to backing store later
      }
    }
    frameNo=(frameNo+1)%NFRAMES;
  }
  return SYSERR;
}
```

2 GCA testing code

For comparing performance between FIFO and GCA, I have written another policy test which takes in number of process to run in parallel. And there is another parameter *oldframe* which refers to an old frame to induce more page faults. The core testing snippet is given below:

```
static void do_policy_test_fifo_gca(void) {
  uint32 npages = PAGE_ALLOCATION - 1;
  uint32 nbytes = npages * PAGESIZE;
  uint32 oldframe=1;

  char *mem = vgetmem(nbytes);
  if (mem == (char*) SYSERR) {
    panic("Page Replacement Policy Test failed\n");
    return;
  }
  for (uint32 i = 0; i<npages; i++) {
    uint32 *p = (uint32*)(mem + (i * PAGESIZE));
    //accessing a new page
    uint32 v = get_test_value(p);
    *p = v;
    //accessing an old page
    int j = i % oldframe;
    p = (uint32*)(mem+(j * PAGESIZE));
    v = get_test_value(p);
    *p = v;
    sleepms(20);
  }
  if (vfreemem(mem, nbytes) == SYSERR) {
    panic("Policy Test: vfreemem() failed.\n");
  }
  kprintf("Fifo,gca passed %d\n",get_faults());
}
```

3 GCA performance over FIFO

Table 1 shows comparative performance between FIFO and GCA, when parallel number of processes are 4. Here each process writes on current page and then refers to a previous page (1,2,3,4,5) in the successive instructions inducing more page faults.

Table 1: Page fault count between FIFO and GCA

NPROC	Refered frame	NFRAME	FIF0	GCA
4	1	50	447	431
4	2	50	499	439
4	3	50	528	441
4	4	50	592	525
4	5	50	692	639

FIFO evicts the pages that are in memory for long times even though they may accessed frequently. Whereas GCA tends to keep pages that are frequently accessed thus inducing less page faults as also can be seen from the figure.

4 Hooks to verify my implementation

For testing I have added codes to hooks. It's basically focuses on two things,

- the selected frame should have access bit and dirty bit set to 0
- all frames between last frame to currently selected frame should have either accessed 1 and dirty 0, or if accessed 0 then it should be marked as not dirty.

5 Additional Details

In my implementation of Lab3, I have tracked virtual heap free space using memories of virtual space.

(I have another implemented tracking free space using free memory too, in my code there is a define macro USE_HEAP_TO_TRACK that controls which version to use.)