Lab 3 Answer for GCA policy

Teng Bian

Data structure:

gca_dirtylist[NFRAMES]: temporary record of whether a page table is dirty or not
gca_pin: the next frame that would be tested
gca_curr: the number of frame that is currently tested

Algorithm:

The algorithm below can find out a frame that can be best evicted. To be specific, a page frame that was not accessed or dirty is the best choice, then the accessed but not dirty or dirty but not accessed one, and the last is accessed and dirty page frame. Here it's for the sake of simplification that I choose to put accessed but not dirty and not accessed but dirty pages in the same privilege. It can be further improved to pick the former first and then latter.

```
For (i=0; i < 3; i++){
        For gca_curr from gca_pin to gca_pin +NFRAMES -1:
                gca_curr = (gca_carr % NFRAMES);
                if gca_curr (th) page frame is not accessed or dirty:        // 00 case
                        restore all dirty bit for pages frames if any;
                        return gca_curr (th) page frame;
                else if gca_curr(th) page frame is accessed but not dirty: // 10 case
                        set corresponding access bit to 1;
                else if gca_curr(th) page frame is not accessed but dirty: // 01 case
                        record in gca_dirtylist that this page is dirty;
                        set corresponding dirty bit to 0;
                else                                                                // 11 case
                        record in gca_dirtylist that this page is dirty;
                        set corresponding dirty bit to 0;
```

I tried with given test code, and my gca policy works well when there is only 1 or 2 processes. The code is as attached.

Code:

```
frame_t *get_pageframe_by_gca(){
    intmask mask;
    mask = disable();

    int32 i;
    int32 gca_plus;
    int32 gca_curr;
    uint32 pgaddr, ptnum;// num of entry
    frame_t *pg_frameptr;
    frame_t *pt_frameptr;
    pt_t *ptptr;

    // gca_pin is already the next stop
    for(i=0; i< 3; i++){
        for(gca_plus=0; gca_plus < NFRAMES; gca_plus++){
            gca_curr = (gcapin+gca_plus)%NFRAMES;
            if(frametab[gca_curr].ftype == PG_TYPE){
                pg_frameptr = &frametab[gca_curr];
                pgaddr = vpage_2_vaddr(pg_frameptr->vid);
                pt_frameptr = (frame_t *) pg_frameptr->advisor;
                ptptr = (pt_t *) fid_2_vaddr(pt_frameptr->fid);
                ptnum = (pgaddr << 10) >> 22;
                if( (ptptr[ptnum].pt_acc==0) && (ptptr[ptnum].pt_dirty==0)){ //00
                    gca_dirtylist_reset();
                    gcapin = gca_curr+1;
                    restore(mask);
                    return pg_frameptr;
                }  //10
                else if( (ptptr[ptnum].pt_acc==1) && (ptptr[ptnum].pt_dirty==0)){
                    ptptr[ptnum].pt_acc = 0;
                }  //11
                else if( (ptptr[ptnum].pt_acc==1) && (ptptr[ptnum].pt_dirty==1)){
                    ptptr[ptnum].pt_dirty = 0;
                    gca_dirtylist[gca_curr] = 1;
                }  //01
                else{
                    ptptr[ptnum].pt_dirty = 0;
                    gca_dirtylist[gca_curr] = 1;
                }
            }
        }
    }

    kprintf("Error: gca can not find pageframe in 3 iterations\n");
    restore(mask);
    return (void *)SYSERR;
}
```