Title: FIFO Page Replacement Practical in xv6-public

Objective:

Learn and implement FIFO page replacement in xv6-public kernel and test it using a user program.

Step 1: Modify `proc.h`

In `proc.h`, add the following definitions to track FIFO pages:

```
#define MAX_PAGES 20

struct proc {
    // existing members ...
    int pages[MAX_PAGES]; // FIFO pages
    int page_count;
};
```

Step 2: Modify `vm.c`

Replace your current `vm.c` with the corrected version below, which includes proper page allocation, deallocation, and FIFO tracking:

```
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"
#include "x86.h"
#include "elf.h"

extern char data[];
pde_t *kpgdir;

void seginit(void) {
  struct cpu *c = &cpus[cpuid()];
  c->gdt[SEG_KCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, 0);
  c->gdt[SEG_KDATA] = SEG(STA_W, 0, 0xffffffff, 0);
  c->gdt[SEG_UCODE] = SEG(STA_X|STA_R, 0, 0xffffffff, DPL_USER);
```

```
  c->gdt[SEG_UDATA] = SEG(STA_W, 0, 0xffffffff, DPL_USER);
  lgdt(c->gdt, sizeof(c->gdt));
}

static pte_t* walkpgdir(pde_t *pgdir, const void *va, int alloc);
static int mappages(pde_t *pgdir, void *va, uint size, uint pa, int perm);

// Implement allocuvm with FIFO tracking
int allocuvm(pde_t *pgdir, uint oldsz, uint newsz) {
  char *mem;
  uint a;
  if(newsz >= KERNBASE) return 0;
  if(newsz < oldsz) return oldsz;

  a = PGROUNDUP(oldsz);
  for(; a < newsz; a += PGSIZE) {
    mem = kalloc();
    if(mem == 0) {
      cprintf("allocuvm out of memory\n");
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }
    memset(mem, 0, PGSIZE);
    if(mappages(pgdir, (char*)a, PGSIZE, V2P(mem), PTE_W|PTE_U) < 0) {
      kfree(mem);
      deallocuvm(pgdir, newsz, oldsz);
      return 0;
    }

    struct proc *cur = myproc();
    if(cur->page_count < MAX_PAGES) {
      cur->pages[cur->page_count++] = (int)a;
      cprintf("Allocated page %d at virtual address 0x%x\n", cur->page_count-
1, a);
    } else {
      int evict = cur->pages[0];
      mem = (char*)P2V(PTE_ADDR(*walkpgdir(pgdir, (void*)evict, 0)));
      kfree(mem);
      for(int i = 1; i < MAX_PAGES; i++) cur->pages[i-1] = cur->pages[i];
      cur->pages[MAX_PAGES-1] = (int)a;
      cprintf("Evicted page at 0x%x, allocated new page at 0x%x\n", evict,
a);
    }
  }
  return newsz;
}

// Other vm.c functions (deallocuvm, freevm, copyuvm, etc.) remain same with
correct declarations.
```

Step 3: Modify `mytest.c`

Create or modify `mytest.c` in the `user/` folder to test FIFO:

```
#include "types.h"
#include "user.h"

#define MAX_PAGES 5      // Number of physical pages available
#define TOTAL_ACCESSES 15 // Total virtual pages to access

int main(int argc, char *argv[])
{
    int fifo[MAX_PAGES];     // Stores allocated virtual pages
    int next_to_replace = 0; // FIFO pointer
    int i, j;
    int page;
    int hit, miss;

    // Initialize page table
    for(i = 0; i < MAX_PAGES; i++)
        fifo[i] = -1;

    hit = 0;
    miss = 0;

    printf(1, "Starting FIFO page replacement simulation...\n");

    // Simulated accesses
    int accesses[TOTAL_ACCESSES] = {0,1,2,3,4,1,5,0,6,1,2,7,3,8,4};
```

```
    for(i = 0; i < TOTAL_ACCESSES; i++) {
        page = accesses[i];
        int found = 0;

        // Check if page is already in memory (hit)
        for(j = 0; j < MAX_PAGES; j++) {
            if(fifo[j] == page) {
                found = 1;
                break;
            }
        }

        if(found) {
            hit++;
            printf(1, "Access page %d: HIT\n", page);
        } else {
            miss++;
            printf(1, "Access page %d: MISS, replacing page %d\n", page,
fifo[next_to_replace]);
            fifo[next_to_replace] = page;
            next_to_replace = (next_to_replace + 1) % MAX_PAGES;
        }
    }

    printf(1, "FIFO simulation completed.\n");
    printf(1, "Total hits: %d\n", hit);
    printf(1, "Total misses: %d\n", miss);
```

```
    exit();
}
```

Step 4: Rebuild xv6

```
make clean
make qemu
```

This will compile the kernel with your modified `vm.c`, `proc.h`, and `mytest.c`.

Step 5: Run Test in QEMU

1. Start QEMU:

```
make qemu
```

2. Run your program:

```
$ mytest
```

Modify the mytest.c for different frame size so hit and miss value can change .