#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <pthread.h>

#include <iostream>

#include <fstream>

#include <math.h>

#include <list>

#include <unordered\_map>

#include <vector>

using namespace std;

#define addr\_file "addresses.txt"

#define LENGTH 1000

#define bin\_file "BACKING\_STORE.bin"

struct Frame{

int data[64];

int frameNum;

};

struct Page{

int data[64];

int pageNum;

};

//TLB class using LRU replacement policy

class TLB{

private:

int capacity;

list<pair<int,int> > l;

unordered\_map<int, list<pair<int,int> >::iterator> map;

public:

TLB(int c){

capacity = c;

}

void add(int page\_num, int frameNum){

auto it = map.find(page\_num);

if(it != map.end()){

it->second->second = frameNum;

l.splice(l.begin(), l, it->second);

}

if(map.size() == capacity){

map.erase(l.back().first);

l.pop\_back();

}

l.push\_front(make\_pair(page\_num, frameNum));

map[page\_num] = l.begin();

//once added, print message like "frame 37 containing page 129 is stored in entry 0 of the TLB"

printf("Frame %d containing page %d is stored in entry %d of the TLB\n\n", frameNum, page\_num, 0);

}

int get(int page\_num){

auto it = map.find(page\_num);

if(it==map.end()) {

//if TLB miss, print message like "frame number for page 129 is missing in the TLB";

printf("Frame number for page %d is missing in the TLB\n", page\_num);

return -1;

}

//if TLB hit, print message like "page 129 is stored in frame 37 which is stored in entry 4 of the TLB";

int entry=0;

std::list<pair<int, int> >::const\_iterator iterator;

for (iterator = l.begin(); iterator != l.end(); ++iterator) {

if(iterator == it->second){

break;

}

entry++;

}

printf("Page %d is stored in frame %d which is stored in entry %d of the TLB\n\n", page\_num, it->second->second, entry);

l.splice(l.begin(), l, it->second);

return it->second->second;

}

};

//Occupied Frame Class using LRU replacemnt policy

class Frames{

private:

list<pair<int,Frame> > l;

int capacity;

unordered\_map<int, list<pair<int,Frame> >::iterator> map;

public:

Frames(int c){

capacity = c;

}

void add(int page\_num, Frame frame, int \*pageNum\_erased, int \*frameNum){

auto it = map.find(page\_num);

if(it != map.end()){

it->second->second = frame;

l.splice(l.begin(), l, it->second);

}

if(map.size() == capacity){

frame.frameNum = l.back().second.frameNum;

\*pageNum\_erased = l.back().first;

map.erase(l.back().first);

l.pop\_back();

}

else{

frame.frameNum = map.size();

}

//print message like "page 129 is loaded into frame 37"

printf("Page %d is loaded into frame %d\n", page\_num, frame.frameNum);

\*frameNum = frame.frameNum;

l.push\_front(make\_pair(page\_num, frame));

map[page\_num] = l.begin();

}

int get(int page\_num){

auto it = map.find(page\_num);

if(it==map.end()) return -1;

l.splice(l.begin(), l, it->second);

return it->second->second.frameNum;

}

};

//Store the 1000 addresses;

vector<int> addr(LENGTH,-1);

//Page table structure;

vector<int> page\_table(16, -1);

//allocate phsical memory

Frames memory = Frames(8);

//allocate TLB space

TLB tlb = TLB(4);

//tlb hit rate

int tlb\_hit = 0;

//page fault rate

int page\_fault = 0;

void getAddress(){

//reading addresses

ifstream afile(addr\_file);

if(afile != NULL){

int i=0; string s;

for(i=0;getline(afile,s)&&i<LENGTH; i++){

addr[i] = atoi(s.c\_str());

int tmp = pow(2,12)-1;

//Becasue Professor modified the requirements, we have to get the least significant 12 bits in address;

addr[i] &= tmp;

}

}

else printf("Failed to open file!\n");

}

int lookIntoPageTable(int address){

//retrive the first 4 bits from 12 bits address;

int first\_4 = (15<<8);

int page\_num = ((address&first\_4)>>8);

//retrive the last 8 bits from 12 bits address;

int last\_8 = pow(2,8)-1;

int offset = address&last\_8;

if(-1 == page\_table[page\_num]){

//page fault;

//print message like "virtual address 33153 contained in page 129 causes a page fault"

printf("Virtual address %d contained in page %d causes a page fault\n", address, page\_num);

page\_fault ++;

return -1;

}

//if found, print like "page 129 is contained in frame 37";

printf("Page %d is contained in frame %d\n\n", page\_num, page\_table[page\_num]);

int frameNum = memory.get(page\_num);

tlb.add(page\_num, frameNum);

return 0;

}

int lookIntoTLB(int address){

//retrive the first 4 bits from 12 bits address;

int first\_4 = (15<<8);

int page\_num = ((address&first\_4)>>8);

//retrive the last 8 bits from 12 bits address;

int last\_8 = pow(2,8)-1;

int offset = address&last\_8;

if(-1 == tlb.get(page\_num))

return -1;

tlb\_hit++;

memory.get(page\_num);

return 0;

}

int load\_page(int address){

//retrive the first 4 bits from 12 bits address;

int first\_4 = (15<<8);

int page\_num = ((address&first\_4)>>8);

//retrive the last 8 bits from 12 bits address;

int last\_8 = pow(2,8)-1;

int offset = address&last\_8;

//read from bin;

Frame frame;

//put frame into memory

int pageNum\_erased = -1;

int frameNum = -1;

memory.add(page\_num, frame, &pageNum\_erased, &frameNum);

tlb.add(page\_num, frameNum);

if(-1 != pageNum\_erased){

//some frame is erased from the memory, we have to erase that from the page\_table too;

page\_table[pageNum\_erased] = -1;

}

page\_table[page\_num] = frameNum;

return 0;

}

int main(){

//read the addresses from the file;

getAddress();

//loop through the array of addresses

for(int i=0; i<LENGTH; i++){

if(-1 == lookIntoTLB(addr[i])){

if(-1 == lookIntoPageTable(addr[i])){

//encountered a page fault, load data from .bin file

load\_page(addr[i]);

}

}

}

printf("\t----- Results -----\n");

//we get all the frames into the vector;

vector<int> all\_frames(8,-1);

printf("Contents of the page table:\n");

for(int i=0; i<16; i++){

if(page\_table[i] == -1)

printf("\tpage %d: not in memory\n", i);

else{

printf("\tpage %d: frame %d\n", i, page\_table[i]);

all\_frames[page\_table[i]] = i;

}

}

printf("\nContents of page frames:\n");

for(int i=0; i<all\_frames.size(); i++){

if(all\_frames[i] > -1)

printf("\tFrame %d: page %d\n", i, all\_frames[i]);

else

printf("\tFrame %d: empty\n", i);

}

printf("\n%d page faults out of 1000 references\n", page\_fault);

printf("\n%d tlb hits out of 1000 references\n", tlb\_hit);

return 0;

}