

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО**

**ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«Донской государственный технический университет»**

**(ДГТУ)**

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

**ЛАБОРАТОРНАЯ РАБОТА № 2**

по дисциплине «Операционные системы»

тема: «МОДЕЛИРОВАНИЕ СТРАНИЧНОЙ ВИРТУАЛЬНОЙ ПАМЯТИ И

АЛГОРИТМОВ СВОПИНГА»

Выполнил: Д. Д. Бачурин

Проверил: Н.В. Кудинов

Ростов-на-Дону

2019

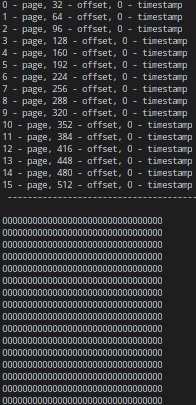
**Цель работы:**

Написать программу, реализующую, согласно варианту (табл. 2), один из алгоритмов выгрузки страниц. В программе должны присутствовать два глобальных массива, один из которых олицетворяет оперативную память, а другой, – внешний носитель. Размер каждого из массивов должен быть не менее чем 256 байт. Использование других глобальных переменных в программе запрещено (то есть вся информация о местоположении страницы памяти и других ее характеристиках должна находиться в массиве «оперативной памяти»). Размер страницы для всех вариантов равен 32 байтам. Программа должна реализовывать сквозную адресацию ячеек «виртуальной памяти». Обязательными для реализации являются следующие функции:

* Чтение ячейки памяти. В результате операции чтения на экран должно быть выдано значение, хранящееся в ячейке памяти. Считать, что в начале работы программы все ячейки памяти равны нулю. В случае если страница с запрошенной ячейкой, расположена на «внешнем носителе», программа должна, используя соответствующий алгоритм, произвести обмен страниц между «оперативной памятью» и «внешним носителем», сообщив об этом пользователю.
* Запись в ячейку памяти. В результате операции, значение ячейки должно быть изменено на введенное пользователем. В случае если страница с запрошенной ячейкой, расположена на «внешнем носителе», программа должна, используя соответствующий алгоритм, произвести обмен страниц между «оперативной памятью» и «внешним носителем», сообщив об этом пользователю.
* Отображение карты распределения страниц виртуальной памяти между «оперативной памятью» и «внешним носителем» (где и в какой позиции расположена каждая из страниц).

**Ход выполнения работы(Алгортим LRU):**

Информация в начале работы.



Фрагмент кода-теста:

printInfo(&mut ram, &mut hdd, start);

write(&mut ram, &mut hdd, 9, &mut [5,3,1], start);

printInfo(&mut ram, &mut hdd, start);

write(&mut ram, &mut hdd, 0, &mut [9,7,6], start);

printInfo(&mut ram, &mut hdd, start);

write(&mut ram, &mut hdd, 11, &mut [2,4,8], start);

printInfo(&mut ram, &mut hdd, start);

read(&mut ram, &mut hdd, 0, start);

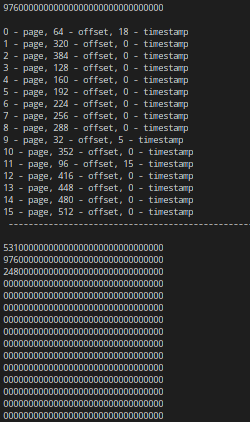
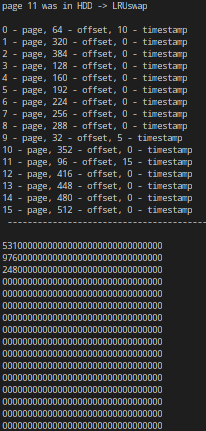
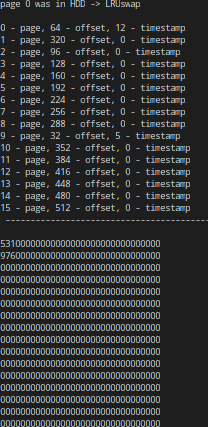
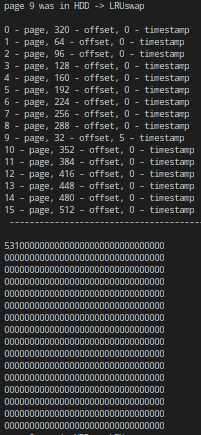
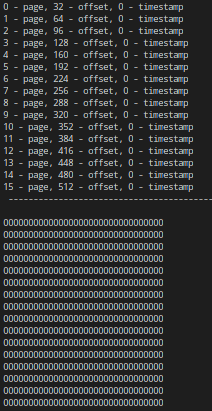
printInfo(&mut ram, &mut hdd, start);

printInfo — вывод информации в терминал (включает в себя смещения offset и временные метки timestamp для каждой из страниц, которые обновляются при записи/чтении страницы.

Write — запись в страницу(3 аргумент — номер страницы, 4 — массив который необходимо записать).

Read — чтение страницы (3 аргумент — номер страницы).

Результат:



Скриншоты читать слева направо, сверху вниз.

На последнем скриншоте в самом верху результат read страницы 0,

т.к она лежала в RAM, то никаких swap-ов не было произведено и не вывелось сообщение.

В архиве лежит файл с page.rs. Исходный код также представлен на следующей странице.

**Код программы:**

use std::time::{Duration, Instant}; //for timestamps

use std::ptr; //for swaps at info seg & data seg&

use std::thread;

//LRU swap LAB2 OS

fn main() {

let start : Instant = Instant::now(); //

const PAGE\_SIZE: usize = 32;

const PAGES: usize = 16;

const PAGE\_INFO\_SIZE: usize = 2;

//offsets for PAGE.info

const PAGE\_OFFSET: usize = 0;

const PAGE\_TIMESTAMP: usize = 1;

const RAM\_SIZE: usize = PAGES \* ( PAGE\_INFO\_SIZE + PAGE\_SIZE / 2);//256 + PAGE\_INFO\_SIZE \* PAGES

const HDD\_SIZE: usize = PAGES \* PAGE\_SIZE / 2; //256

const RAM\_PAGES: usize = (RAM\_SIZE - INFO\_OFFSET)/PAGE\_SIZE; //8

const HDD\_PAGES: usize = HDD\_SIZE/PAGE\_SIZE; //8

const INFO\_OFFSET: usize = PAGES \* PAGE\_INFO\_SIZE; //32

let mut ram: [usize; RAM\_SIZE] = [0; RAM\_SIZE];

let mut hdd: [usize; HDD\_SIZE] = [0; HDD\_SIZE];

let mut offset: i32 = 0;

fn LRUswap(ram: &mut [usize], hdd: &mut [usize], x: usize) {

//find latest used page in RAM!

//then swap page data + info with x data + info

let mut min\_timestamp: usize = std::usize::MAX;

let mut page: usize = 0;

for i in 0..(INFO\_OFFSET/PAGE\_INFO\_SIZE) {

if (ram[i \* PAGE\_INFO\_SIZE + PAGE\_OFFSET] < RAM\_SIZE

&& ram[i \* PAGE\_INFO\_SIZE + PAGE\_TIMESTAMP] < min\_timestamp) {

page = i;

min\_timestamp = ram[i \* PAGE\_INFO\_SIZE + PAGE\_TIMESTAMP];

}

}

//ptrs data&info

let x\_offset : usize = ram[x \* PAGE\_INFO\_SIZE + PAGE\_OFFSET]; //page\_data\_offset(ram, x);

let page\_offset : usize = ram[page \* PAGE\_INFO\_SIZE + PAGE\_OFFSET]; //page\_data\_offset(ram, page);

let page\_info\_ptr = ram[page \* PAGE\_INFO\_SIZE..].as\_mut\_ptr() as \*mut [usize; PAGE\_INFO\_SIZE];

let x\_info\_ptr = ram[x \* PAGE\_INFO\_SIZE..].as\_mut\_ptr() as \*mut [usize; PAGE\_INFO\_SIZE];

let x\_data\_ptr = hdd[x\_offset - RAM\_SIZE..].as\_mut\_ptr() as \*mut [usize; PAGE\_SIZE];

let page\_data\_ptr = ram[page\_offset..].as\_mut\_ptr() as \*mut [usize; PAGE\_SIZE];

//swap info & data for x & page

unsafe {

ptr::swap(x\_data\_ptr, page\_data\_ptr);

ptr::swap(x\_info\_ptr, page\_info\_ptr);

}

}

//if x in HDD then swap and write to x

fn write(ram: &mut [usize], hdd: &mut [usize], x: usize, data: &[usize], start : Instant) {

let mut x\_offset: usize = ram[x \* PAGE\_INFO\_SIZE + PAGE\_OFFSET];

if (ram[x \* PAGE\_INFO\_SIZE + PAGE\_OFFSET] >= RAM\_SIZE) {

LRUswap(ram, hdd, x);

print!("\npage {} was in HDD -> LRUswap\n", x);

x\_offset = ram[x \* PAGE\_INFO\_SIZE + PAGE\_OFFSET];

}

thread::sleep(Duration::from\_millis(5)); //sleep 2 ms

let now = Instant::now();

ram[x \* PAGE\_INFO\_SIZE + PAGE\_TIMESTAMP] = now.duration\_since(start).subsec\_millis() as usize;

for i in 0..data.len() {

ram[x\_offset + i] = data[i];

}

}

//if x in HDD then swap and write to x

fn read(ram: &mut [usize], hdd: &mut [usize], x: usize, start : Instant) {

let mut x\_offset: usize = ram[x \* PAGE\_INFO\_SIZE + PAGE\_OFFSET];

if (ram[x \* PAGE\_INFO\_SIZE + PAGE\_OFFSET] >= RAM\_SIZE) {

LRUswap(ram, hdd, x);

print!("\npage {} was in HDD -> LRUswap\n", x);

x\_offset = ram[x \* PAGE\_INFO\_SIZE + PAGE\_OFFSET];

}

thread::sleep(Duration::from\_millis(2)); //sleep 2 ms

let now = Instant::now();

ram[x \* PAGE\_INFO\_SIZE + PAGE\_TIMESTAMP] = now.duration\_since(start).subsec\_millis() as usize;

print!("\n\n");

for i in 0..PAGE\_SIZE {

print!("{}", ram[x\_offset + i])

}

print!("\n");

}

fn printInfo(ram: &mut [usize], hdd: &mut [usize], start : Instant) {

print!("\n");

for i in 0..PAGES {

print!("{:?} - page, {:?} - offset, {:?} - timestamp\n", i

, ram[i \* PAGE\_INFO\_SIZE + PAGE\_OFFSET]

, ram[i \* PAGE\_INFO\_SIZE + PAGE\_TIMESTAMP]);

}

print!("{:-<1$}\n", " ", 50);

for i in 0..RAM\_PAGES {

print!("\n");

for j in 0..PAGE\_SIZE {

print!("{}", ram[INFO\_OFFSET + j + PAGE\_SIZE \* i])

}

}

for i in 0..HDD\_PAGES {

print!("\n");

for j in 0..PAGE\_SIZE {

print!("{}", hdd[j + PAGE\_SIZE \* i])

}

}

/\*for i in 0..PAGES {

read(ram, hdd, i, start);

print!("\n");

}\*/

}

//init : set offsets for every page

for i in 0..PAGES {

ram[i \* PAGE\_INFO\_SIZE + PAGE\_OFFSET] = (INFO\_OFFSET + i \* PAGE\_SIZE);

}

printInfo(&mut ram, &mut hdd, start);

write(&mut ram, &mut hdd, 9, &mut [5,3,1], start);

printInfo(&mut ram, &mut hdd, start);

write(&mut ram, &mut hdd, 0, &mut [9,7,6], start);

printInfo(&mut ram, &mut hdd, start);

write(&mut ram, &mut hdd, 11, &mut [2,4,8], start);

printInfo(&mut ram, &mut hdd, start);

read(&mut ram, &mut hdd, 0, start);

printInfo(&mut ram, &mut hdd, start);

}

**Вывод:** в ходе работы реализовал алгоритм свопинга страниц LRU и опробовал на 16 страницах(8 ram, 8 hdd).