МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра обчислювальної техніки

**ЛАБОРАТОРНА РОБОТА 2**

з дисципліни

“Операційні Системи”

Алгоритмы замещения страниц виртуальной памяти

Студента 3 курсу

групи ІП-64

Вінницький В.А.

Перевірив: Сімоненко А.В.

Київ 2018

**Завдання:**

Розробити модель системи з сторінкової організацією пам'яті,

задовольняє наведеним вище вихідними даними, з реалізацією одного

з алгоритмів заміщення сторінок.

1. NRU, що не використовувалася останнім часом сторінка.

2. Алгоритм годинник, модифікований алгоритм друга спроба.

3. Алгоритм aging , модифікований NFU.

4. Алгоритм робочий набір.

5. Алгоритм WSClock.

Політику розподілу пам'яті (локальна або глобальна) вибрати самостійно.

**Опис лабораторної роботи:**

В системі є N сторінок фізичної пам'яті. Кожній віртуальної сторінці

відповідає рівно одна фізична сторінка.

В системі є кілька процесів. Кожен процес має власне ВАП(virtual address prostir), число сторінок в якому може бути як менше, так і більше ніж N. Будь-яка фізична сторінка може бути відображена ВАП тільки одного процесу.

Кожен процес періодично звертається до якоїсь своїй сторінці. Це може  
бути як нова сторінка, або та до якої було звернення в минулий раз, але всі процеси характеризуються локальностью звернень (90/10).

Звернення може бути читанням або записом (50/50).

В системі використовується один з алгоритмів заміщення сторінок. Сторінки заміщуються на вимогу (demand paging). В системі є таймер і по  
необхідності можуть бути запущені додаткові системні процеси,   
наприклад, фоновий процес для різних перевірок стану пам'яті.

**Лістинг:**

==> lab2.h <==

#ifndef PAGESIM\_H

#define PAGESIM\_H

// List for page tables and victim lists

LIST\_HEAD(Page\_Ref\_List, Page\_Ref) page\_refs;

// List for page tables and victim lists

LIST\_HEAD(Frame\_List, Frame);

// stuct to hold Frame info

typedef struct Page\_Ref

{

LIST\_ENTRY(Page\_Ref) pages;

int page\_num;

} Page\_Ref;

typedef struct Frame

{

LIST\_ENTRY(Frame) frames;

int index;

int page; // page frame points to, -1 is empty

int r\_bit; // R bit

} Frame;

typedef struct {

int hits; // number of times page was found in page table

int misses; // number of times page wasn't found in page table

struct Frame\_List page\_table; // List to hold frames in page table

struct Frame\_List victim\_list; // List to hold frames that were replaced in page table

Frame \*last\_victim; // Holds last frame used as a victim to make inserting to victim list faster

} Clock\_Data;

void gen\_page\_refs();

Page\_Ref\* gen\_ref();

Clock\_Data \*create\_algo\_data\_store(); // returns empty algorithm data

Frame \*create\_empty\_frame(int index);

int cleanup();

int event\_loop();

int page(int page\_ref); // page with page ref

int get\_ref(); // get next page ref however you like

int add\_victim(struct Frame\_List \*victim\_list, struct Frame \*frame); // add victim frame to a victim list

int print\_list(struct Frame \*head, const char\* index\_label, const char\* value\_label);

int print\_stats();

int print\_summary();

int CLOCK(Clock\_Data \*data);

#endif

==> lab2.c <==

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <time.h>

#include <sys/queue.h>

#include "lab2.h"

int num\_frames = 10; // Number of avaliable pages in page tables

int page\_ref\_upper\_bound = 12; // Largest page reference

int max\_page\_calls = 1000; // Max number of page refs to test

int last\_page\_ref = -1; // Last ref

Clock\_Data \*clock\_data;

int main()

{

clock\_data = create\_algo\_data\_store();

gen\_page\_refs();

event\_loop();

cleanup();

return 0;

}

void gen\_page\_refs()

{

int num\_refs = 0;

LIST\_INIT(&page\_refs);

Page\_Ref \*page = gen\_ref();

LIST\_INSERT\_HEAD(&page\_refs, page, pages);

while(num\_refs < max\_page\_calls)

{ // generate a page ref up too max\_page\_calls and add to list

LIST\_INSERT\_AFTER(page, gen\_ref(), pages);

page = page->pages.le\_next;

num\_refs++;

}

return;

}

Page\_Ref\* gen\_ref()

{

Page\_Ref \*page = malloc(sizeof(Page\_Ref));

page->page\_num = rand() % page\_ref\_upper\_bound;

return page;

}

Clock\_Data \*create\_algo\_data\_store()

{

Clock\_Data \*data = malloc(sizeof(Clock\_Data));

data->hits = 0;

data->misses = 0;

data->last\_victim = NULL;

/\* Initialize Lists \*/

LIST\_INIT(&(data->page\_table));

LIST\_INIT(&(data->victim\_list));

/\* Insert at the page\_table. \*/

Frame \*framep = create\_empty\_frame(0);

LIST\_INSERT\_HEAD(&(data->page\_table), framep, frames);

/\* Build the rest of the list. \*/

size\_t i = 0;

for (i = 1; i < num\_frames; ++i)

{

LIST\_INSERT\_AFTER(framep, create\_empty\_frame(i), frames);

framep = framep->frames.le\_next;

}

return data;

}

Frame\* create\_empty\_frame(int index)

{

Frame \*framep = malloc(sizeof(Frame));

framep->index = index;

framep->page = -1;

framep->r\_bit = 0;

return framep;

}

int event\_loop()

{

int counter = 0;

while(counter < max\_page\_calls)

{

page(get\_ref());

++counter;

}

print\_summary();

return 0;

}

int get\_ref()

{

if (page\_refs.lh\_first != NULL)

{ // pop Page\_Ref off page\_refs

int page\_num = page\_refs.lh\_first->page\_num;

LIST\_REMOVE(page\_refs.lh\_first, pages);

return page\_num;

}

else

{

return rand() % page\_ref\_upper\_bound;

}

}

int page(int page\_ref)

{

last\_page\_ref = page\_ref;

CLOCK(clock\_data);

print\_stats();

return 0;

}

int add\_victim(struct Frame\_List \*victim\_list, struct Frame \*frame)

{

printf("Victim index: %d, Page: %d\n", frame->index, frame->page);

struct Frame \*victim = malloc(sizeof(Frame));

\*victim = \*frame;

victim->index = 1;

LIST\_INSERT\_HEAD(victim\_list, victim, frames);

return 0;

}

int CLOCK(Clock\_Data \*data)

{

static Frame \*clock\_hand = NULL;

Frame \*framep = data->page\_table.lh\_first;

int fault = 0;

/\* Find target (hit), empty page slot (miss), or victim to evict (miss) \*/

while(framep != NULL && framep->page > -1 && framep->page != last\_page\_ref)

framep = framep->frames.le\_next;

if(framep != NULL)

{

if(framep->page == -1)

{

framep->page = last\_page\_ref;

framep->r\_bit = 0;

fault = 1;

}

else

{ // Found the page, update its R bit to 0

framep->r\_bit = 0;

}

}

else

{

while(clock\_hand == NULL || clock\_hand->r\_bit == 0)

{

if(clock\_hand == NULL)

{

clock\_hand = data->page\_table.lh\_first;

}

else

{

clock\_hand->r\_bit = 1;

clock\_hand = clock\_hand->frames.le\_next;

}

}

add\_victim(&data->victim\_list, clock\_hand);

clock\_hand->page = last\_page\_ref;

clock\_hand->r\_bit = 0;

fault = 1;

}

if(fault == 1) data->misses++; else data->hits++;

return fault;

}

int print\_stats()

{

print\_summary();

print\_list(clock\_data->page\_table.lh\_first, "Frame #", "Page Ref");

return 0;

}

int print\_summary()

{

printf("Frames in Mem: %d, ", num\_frames);

printf("Hits: %d, ", clock\_data->hits);

printf("Misses: %d, ", clock\_data->misses);

printf("Hit Ratio: %f\n", (double)clock\_data->hits/(double)(clock\_data->hits+clock\_data->misses));

return 0;

}

int print\_list(struct Frame \*head, const char\* index\_label, const char\* value\_label)

{

int colsize = 9, labelsize;

struct Frame \*framep;

// Determine lanbel col size from text

if (strlen(value\_label) > strlen(index\_label))

labelsize = strlen(value\_label) + 1;

else

labelsize = strlen(index\_label) + 1;

/\* Forward traversal. \*/

printf("%-\*s: ", labelsize, index\_label);

for (framep = head; framep != NULL; framep = framep->frames.le\_next)

{

printf("%\*d", colsize, framep->index);

}

printf("\n%-\*s: ", labelsize, value\_label);

for (framep = head; framep != NULL; framep = framep->frames.le\_next)

{

if(framep->page == -1)

printf("%\*s", colsize, "\_");

else

printf("%\*d", colsize, framep->page);

}

printf("\n%-\*s: ", labelsize, "R bit");

for (framep = head; framep != NULL; framep = framep->frames.le\_next)

{

printf("%\*d", colsize, framep->r\_bit);

}

printf("\n\n");

return 0;

}

int cleanup()

{

while (clock\_data->page\_table.lh\_first != NULL)

{

LIST\_REMOVE(clock\_data->page\_table.lh\_first, frames);

}

while (clock\_data->victim\_list.lh\_first != NULL)

{

LIST\_REMOVE(clock\_data->victim\_list.lh\_first, frames);

}

return 0;

}