Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Системне програмне забезпечення – 2

Лабораторна робота №1

**«Алгоритми заміщення сторінок віртуальної пам’яті»**

Виконала:

студентка групи ІВ-71

Молчанова В.С.

Перевірив:

ст. вик. Сімоненко А.В.

Київ

2020 р.

### Завдання

Номер залікової книжки: 7110

Варіант завдання: алгоритм NRU, не использовавшаяся в последнее время страница

### Опис алгоритму

Алгоритм NRU (Not Recently Used) видаляє сторінку за допомогою випадкового пошуку в не порожньому класі з найменшим номером. Коли процес запускається, обидва сторінкових біта (R і M) для всіх його сторінок операційною системою скинуті в 0. Періодично (наприклад, при кожному перериванні по таймеру) біт R очищається з метою відрізнити сторінки, до яких давно не було звернення, від тих, на які посилання були.

При порушенні сторінкового переривання операційна система перевіряє всі сторінки і ділить їх на чотири категорії на підставі поточних значень бітів R і М:

* клас 0: не було звернень і змін (R = 0, M = 0);
* клас 1: не було звернень, сторінка змінена (R = 0, M = 1);
* клас 2: було звернення, сторінка не змінена (R = 1, M = 0);
* клас 3: сталося і звернення, і зміна (R = 1, M = 1).

### Структури даних

*Class PhysicalPage* – опис фізичної сторінки з полем *data* що містить дані, які пізніше потраплять до свопу.

*Class VirtualPage* – опис віртуальної сторінки з полями R, M, present, ppn (номер відповідної фізичної сторінки)

*Class VirtualMemory* – таблиця, що відображає логічні (віртуальні) адреси до

фізичного простору (якщо таке відображення присутнє).

*Class Process* – клас процесу в системі, що має власні id та віртуальний простір пам’яті.

*Class Os* – клас для виконання симуляції та зберігання часу, процесів, статистики

### Лістинг програми

Main.kt

const val PHYS\_PAGES = 5

const val CLEAR\_TIME = 10

const val TOTAL\_TIME = 50

const val PROCESSES = 2

fun main(args: Array<String>){

val os = Os()

os.go()

}

val Boolean.int

get() = if (this) 1 else 0

val Int.bool

get() = this == 1

Os.kt

import kotlin.math.roundToInt

class Os {

private val processes = List(PROCESSES) {createProcess(it)}

private val physicalPages = List(PHYS\_PAGES){PhysicalPage()}

private val swap = mutableListOf<String>()

private var allocatedPages: Int = 0

private var clock: Int = 1

private var pageFaults: Int = 0

private var pageSuccesses: Int = 0

fun go() {

println("==== OS CONFIGURATION ====")

println("$PHYS\_PAGES physical pages")

println("$PROCESSES processes:")

for (process in processes)

{

println("\t\* p${process.id} with ${process.vm.pages.count()} virtual pages")

}

println()

println("===== SIMULATION LOG =====")

for (i in 1..TOTAL\_TIME) {

val currentProcess = processes[i % PROCESSES]

val demandedPage = currentProcess.demandPage()

val accessed = tryAccess(currentProcess.id, demandedPage, (0..1).random().bool)

if (accessed == null) println ("UNEXPECTED ERROR")

clock++

if (clock % CLEAR\_TIME == 0) clearR()

}

println()

println("======= STATISTICS =======")

println("Page faults: $pageFaults")

println("Page fault ratio: ${(pageFaults/TOTAL\_TIME.toDouble()\*100).roundToInt()}%")

}

private fun tryAccess(pid: Int, pageIdx:Int, write: Boolean): Int? {

val procIdx = processes.indexOfFirst { p -> p.id == pid }

// Check if the demanded page is present

val direct = tryTranslate(procIdx,pageIdx, write)

if (direct != null) return direct

// Check if there're any free physical pages

val allocated = tryAllocate(procIdx, pageIdx, write)

if (allocated != null) return allocated

// Find page by the NRU algorithm

return tryReplace(procIdx, pageIdx, write)

}

private fun tryTranslate(procIdx: Int, pageIdx:Int, write: Boolean): Int?{

val process = processes[procIdx]

val pid = process.id

val page = process.vm.pages[pageIdx]

return if (!page.present) {

println("$clock. Page miss for p$pid: $pageIdx -> ???")

pageFaults++

null

} else {

println("$clock.Page hit for p$pid: $pageIdx -> ${page.ppn}")

pageSuccesses++

page.R = true

if (write) {

page.M = true

}

page.ppn

}

}

private fun tryAllocate(procIdx: Int, pageIdx:Int, write: Boolean): Int? {

return if (allocatedPages < PHYS\_PAGES) {

val process = processes[procIdx]

val pid = process.id

mapPage(procIdx, pageIdx, write, allocatedPages)

println("\tAllocated page for p$pid: $pageIdx -> $allocatedPages")

allocatedPages++

} else null

}

private fun tryReplace(procIdx: Int, pageIdx:Int, write: Boolean): Int? {

val process = processes[procIdx]

val pid = process.id

for (i in 0 until 4) {

val oldPageIdx = process.vm.getPageOfClass(i)?: continue

val oldPage = process.vm.pages[oldPageIdx]

val ppn = oldPage.ppn

// Move the data of the less used page to the swap

swap.add(physicalPages[ppn].data)

oldPage.present = false

oldPage.ppn = -1

println("\tEvicted page $ppn of class $i, owned by p$pid page $oldPageIdx")

mapPage(procIdx, pageIdx, write, ppn)

println("\t\tMapped page for p$pid: $pageIdx -> $ppn")

return ppn

}

return null

}

private fun mapPage(procIdx: Int, pageIdx:Int, write: Boolean, dst:Int) {

val page = processes[procIdx].vm.pages[pageIdx]

page.present = true

page.ppn = dst

page.R = true

if (write) {

page.M = true

}

}

private fun clearR() {

for (process in processes) {

process.vm.clearR()

}

println("CLEARED BIT R")

}

private fun createProcess(id: Int): Process {return Process(id, (PHYS\_PAGES - 2..PHYS\_PAGES + 2).random())}

}

Process.kt

class Process(val id: Int, pagesCount: Int) {

val vm = VirtualMemory(pagesCount)

fun demandPage(): Int {return (0 until vm.pages.count()).random()}

}

VirtualMemory.kt

class VirtualMemory(pagesCount: Int) {

val pages = List(pagesCount) {VirtualPage()}

fun clearR() {

for (page in pages) {

page.R = false

}

}

fun getPageOfClass(n: Int): Int? {

val pagesOfClass = mutableListOf<Int>()

for (i in 0 until pages.count()) {

val page = pages[i]

if ( page.present && page.state == n ) pagesOfClass.add(i)

}

return if (pagesOfClass.isEmpty()) null else pagesOfClass.random()

}

}

VirtualPage.kt

class VirtualPage {

var present: Boolean = false

// Physical page number

var ppn: Int = -1

var R: Boolean = false

var M: Boolean = false

val state: Int

get() = R.int \* 2 + M.int

}

PhysicalPage.kt

class PhysicalPage {

var count: Int = 0

val data: String

get() = "Data #${count++}"

}

### Результат виконання

==== OS CONFIGURATION ====

5 physical pages

2 processes:

\* p0 with 7 virtual pages

\* p1 with 3 virtual pages

===== SIMULATION LOG =====

1. Page miss for p1: 0 -> ???

Allocated page for p1: 0 -> 0

2. Page miss for p0: 2 -> ???

Allocated page for p0: 2 -> 1

3. Page miss for p1: 1 -> ???

Allocated page for p1: 1 -> 2

4.Page hit for p0: 2 -> 1

5.Page hit for p1: 0 -> 0

6. Page miss for p0: 3 -> ???

Allocated page for p0: 3 -> 3

7. Page miss for p1: 2 -> ???

Allocated page for p1: 2 -> 4

8. Page miss for p0: 6 -> ???

Evicted page 3 of class 2, owned by p0 page 3

Mapped page for p0: 6 -> 3

9.Page hit for p1: 2 -> 4

CLEARED BIT R

10. Page miss for p0: 3 -> ???

Evicted page 3 of class 0, owned by p0 page 6

Mapped page for p0: 3 -> 3

11.Page hit for p1: 1 -> 2

12.Page hit for p0: 3 -> 3

13.Page hit for p1: 1 -> 2

14. Page miss for p0: 6 -> ???

Evicted page 1 of class 1, owned by p0 page 2

Mapped page for p0: 6 -> 1

15.Page hit for p1: 2 -> 4

16. Page miss for p0: 4 -> ???

Evicted page 1 of class 2, owned by p0 page 6

Mapped page for p0: 4 -> 1

17.Page hit for p1: 0 -> 0

18.Page hit for p0: 3 -> 3

19.Page hit for p1: 2 -> 4

CLEARED BIT R

20. Page miss for p0: 1 -> ???

Evicted page 3 of class 1, owned by p0 page 3

Mapped page for p0: 1 -> 3

21.Page hit for p1: 2 -> 4

22. Page miss for p0: 2 -> ???

Evicted page 1 of class 1, owned by p0 page 4

Mapped page for p0: 2 -> 1

23.Page hit for p1: 2 -> 4

24.Page hit for p0: 2 -> 1

25.Page hit for p1: 2 -> 4

26. Page miss for p0: 3 -> ???

Evicted page 1 of class 3, owned by p0 page 2

Mapped page for p0: 3 -> 1

27.Page hit for p1: 0 -> 0

28. Page miss for p0: 4 -> ???

Evicted page 1 of class 3, owned by p0 page 3

Mapped page for p0: 4 -> 1

29.Page hit for p1: 1 -> 2

CLEARED BIT R

30. Page miss for p0: 5 -> ???

Evicted page 3 of class 1, owned by p0 page 1

Mapped page for p0: 5 -> 3

31.Page hit for p1: 1 -> 2

32. Page miss for p0: 3 -> ???

Evicted page 1 of class 1, owned by p0 page 4

Mapped page for p0: 3 -> 1

33.Page hit for p1: 2 -> 4

34. Page miss for p0: 4 -> ???

Evicted page 3 of class 2, owned by p0 page 5

Mapped page for p0: 4 -> 3

35.Page hit for p1: 1 -> 2

36. Page miss for p0: 5 -> ???

Evicted page 1 of class 3, owned by p0 page 3

Mapped page for p0: 5 -> 1

37.Page hit for p1: 2 -> 4

38. Page miss for p0: 2 -> ???

Evicted page 1 of class 2, owned by p0 page 5

Mapped page for p0: 2 -> 1

39.Page hit for p1: 0 -> 0

CLEARED BIT R

40. Page miss for p0: 0 -> ???

Evicted page 3 of class 1, owned by p0 page 4

Mapped page for p0: 0 -> 3

41.Page hit for p1: 2 -> 4

42.Page hit for p0: 2 -> 1

43.Page hit for p1: 2 -> 4

44. Page miss for p0: 3 -> ???

Evicted page 3 of class 2, owned by p0 page 0

Mapped page for p0: 3 -> 3

45.Page hit for p1: 2 -> 4

46. Page miss for p0: 4 -> ???

Evicted page 1 of class 3, owned by p0 page 2

Mapped page for p0: 4 -> 1

47.Page hit for p1: 0 -> 0

48.Page hit for p0: 4 -> 1

49.Page hit for p1: 1 -> 2

CLEARED BIT R

50. Page miss for p0: 0 -> ???

Evicted page 3 of class 1, owned by p0 page 3

Mapped page for p0: 0 -> 3

======= STATISTICS =======

Page faults: 22

Page fault ratio: 44%

### Висновок

Під час виконання лабораторної роботи, було розроблено програму, яка виконує заміщення сторінок згідно алгоритму NRU(Not Recently Used). Використаний алгоритм відрізняється від інших середньою складністю програмування, інтуїтивно зрозумілий для розуміння та дає оптимальну продуктивність.