МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний авіаційний університет

Лабораторна робота № 3

з дисципліни ««Високопродуктивні обчислювальні системи»

Тема: «Засоби синхронізації процесів:монітори»

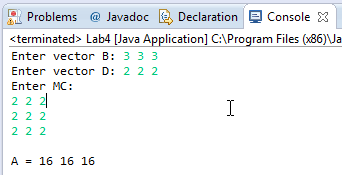
Київ – 2015

**Завдання:**

Розробити програму для ВПОС, яка включає4 процесори і два пристрої введення-виведення. Математична задача ―обчислення виразуA = B\* MC – D. Введення векторів B і D виконується в другому процесорі, матриці МС – в третьому процесорі, виведення результату А – в третьому процесорі.

|  |  |  |  |
| --- | --- | --- | --- |
| **№ варіанту** | Операція | **Номера процесору ВПОС (4 процесори)** | |
| Для введення | Для виведення |
| 25 | A = B\* MC – D | B,D – 2  MC - 3 | 3 |

**Скріншот роботи програми:**



**Лістинг коду:**

*// Файл Monitor.java*

**public class** Monitor {

**protected int** barrier = 2;

**protected boolean** canOutputResult = **false**;

**protected boolean** inputIsFree = **true**;

// метод очікування сигналу вводу

**public synchronized void** waitVarInput() {

**while** (barrier != 0) {

**try** {

wait();

} **catch** (InterruptedException e) {}

}

}

// метод, що сигналізує про введення змінних

**public synchronized void** signalVarInput() {

barrier -= 1;

notify();

}

// метод очікування сигналу виводу результату

**public synchronized void** waitResult() {

**while** (!canOutputResult) {

**try** {

wait();

} **catch** (InterruptedException e) {}

}

}

// метод, що сигналізує про дозвіл виводу результату

**public synchronized void** signalResult() {

canOutputResult = **true**;

notify();

}

// метод, що зобороняє виконання вводу

**public synchronized void** holdInput() {

inputIsFree = **false**;

}

// метод, що очікує на звільнення вводу

**public synchronized void** waitForFreeInput() {

**while** (!inputIsFree) {

**try** {

wait();

} **catch** (InterruptedException e) {}

}

}

// метод, що сигналізує про звільнення вводу

**public synchronized void** signalInputIsFree() {

inputIsFree = **true**;

notify();

}

}

// Файл Lab4.java

**import** java.util.Scanner;

**public class** Lab4 {

// Метод для вводу вектора розміром N

**public static int**[] vectorInput(){

Scannersc = **new** Scanner(System.*in*);

**int**[] A = **new int**[*N*];

**for** (**int** i = 0; i <*N*; i++) {

A[i] = sc.nextInt();

}

**return** A;

}

// Метод для вводу матриці розміром NxN

**public static int**[][] matrixInput(){

Scannersc = **new** Scanner(System.*in*);

**int**[][] MA = **new int**[*N*][*N*];

**for** (**int** i = 0; i <*N*; i++) {

**for** (**int** j = 0; j <*N*; j++) {

MA[i][j] = sc.nextInt();

}

}

**return** MA;

}

// Метод для множення матриці на вектор

**public static int**[] matrixVectorMul(**int** MA[][], **int** A[]){

**int**[] x = **new int**[*N*];

**int**tmp;

**for** (**int** i = 0; i <*N*; i++) {

tmp = 0;

**for** (**int** j = 0; j <*N*; j++) {

tmp = tmp + MA[i][j] \* A[j];

}

x[i] = tmp;

}

**return** x;

}

// Метод для віднімання двох векторів

**public static int**[] vectorSub(**int** a[], **int** b[]) {

**int**[] x = **new int**[*N*];

**for** (**int** i = 0; i <*N*; i++) {

x[i] = a[i] - b[i];

}

**return** x;

}

// Метод для виведення вектора

**public static void** vectorOutput(**int** a[]){

**for** (**int** i = 0; i <*N*; i++) {

System.*out*.print(a[i] + " ");

}

System.*out*.println();

}

// Глобальні змінні АКА загальна пам'ять

**public static int**[] *A*, *B*, *D*;

**public static int**[][] *MC*;

**final static int** *N* = 3; // константа розмірності для матриць та векторів

**public static void** main(String[] args) {

// Процес 2

**class** Thread2 **extends**Thread {

Monitormonitor;

**public** Thread2 (Monitormon) {

monitor = mon;

}

**public void** run(){

monitor.waitForFreeInput();

monitor.holdInput(); // захоплення монітору введення

// Введення векторів B і D

System.*out*.print("Entervector B: ");

*B* = *vectorInput*();

System.*out*.print("Entervector D: ");

*D* = *vectorInput*();

monitor.signalInputIsFree(); // звільнення монітору введення

monitor.signalVarInput(); // сигнал дозволу виконання арифм. операцій

}

}

// Процес 3

**class** Thread3 **extends** Thread {

Monitormonitor;

**public** Thread3 (Monitormon) {

monitor = mon;

}

**public void** run(){

monitor.waitForFreeInput();

monitor.holdInput(); // захолення монітору введення

// введення матриці МС

System.*out*.println("Enter MC: ");

*MC* = *matrixInput*();

monitor.signalInputIsFree(); // сигнал звільнення монітору введення

monitor.signalVarInput(); // сигнал дозволу виконання арифм. операцій

monitor.waitResult(); // очікування сигналу виведення результату

// виведення результату

System.*out*.print("\nA = ");

*vectorOutput*(*A*);

}

}

// Процес 4

**class** Thread4 **extends** Thread {

Monitormonitor;

**public** Thread4 (Monitormon) {

monitor = mon;

}

**public void** run(){

monitor.waitVarInput(); // Очікування двох сигналів дозволу для виконання арифм. операцій

*A* = *vectorSub*(*matrixVectorMul*(*MC*, *B*), *D*); // Обчислення арифм. виразу

monitor.signalResult(); // сигнал дозволу виводу результату

}

}

// Створення об'єкту монітора

Monitor m = **new** Monitor();

// Створення об'єктів тредів

Thread2 T2 = **new** Thread2(m);

Thread3 T3 = **new** Thread3(m);

Thread4 T4 = **new** Thread4(m);

// Запуск тредів

T2.start();

T3.start();

T4.start();

}

}

Для управління паралельним виконанням потоків, зокрема для забезпечення доступу до спільно використовуваних ресурсів , використовується сінхронізація. Механізм синхронізації ґрунтується на концепції монітора.

Монітор ― це об'єкт спеціального призначення, в якому застосований принцип взаємного виключення (mutualexclusion) для груп процедур. Під час виконання програми монітор допускає лише почергове виконання процедури, яка знаходиться під її контролем. У кожного об'єкта в Java є свій власний неявний монітор. Коли метод типу synchronized викликається для об'єкта, відбувається звернення до монітора об'єкта щоб визначити, чи виконує в даний момент будь-якої інший потік метод типу synchronized для даного об'єкта. Якщо немає, то поточний потік отримує дозвіл увійти в монітор. Вхід в монітор називається також блокуванням (locking) монітора. Якщо при цьому інший потік вже увійшов в монітор, то поточний потік повинен чекати доти, доки інший потік не покине монітор.

Монітор Java вводить почерговість в паралельну обробку. Цей спосіб називається також перетворенням в послідовну форму (serialization). Оголошення методу synchronized не має на увазі, що тільки один потік може одночасно виконувати цей метод, як у випадку критичного ділянки (criticalsections). Мається на увазі, що в будь-який момент часу тільки один потік може викликати цей метод (або будь-який інший метод типу synchronized ) для конкретного об'єкта.

Такім чином, монітори Java пов'язані з об'єктами, але не з блоками кода.Два потоку можуть паралельно виконувати один і той же метод типу synchronized за умови, що цей метод викликаний для різних об'єктів.Монітори не є об'єктами мови Java, у них немає атрибутів або методів. Доступ до моніторів можливий на рівні коду JVM.