ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Факультет кибербезопасности и управления

Кафедра программной инженерии

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №** 5

по дисциплине Операционные системы и оболочки

Взаимоблокировка

|  |  |  |
| --- | --- | --- |
| **ВЫПОЛНИЛ** | | |
| студент | РПИС-11  (группа) | Яковлев П. Н.  (ФИО) |
|  | **ПРОВЕРИЛ** |  |
| (должность) | | **Баженов А. Э**.  (ФИО) |

Самара

2023

Цель лабораторной работы – освоить навыки обнаружения взаимоблокировок при наличии нескольких разделяемых ресурсов, а также выход и их предотвращение. Научиться строить граф разделяемых ресурсов.

Краткие теоретические сведения

Взаимоблокировка (тупиковая ситуация, **deadlock**) — это состояние, при котором каждый поток ожидает на освобождение одного из ресурсов, а все ресурсы при этом захвачены. Потоки будут ожидать друг друга, и они никогда не смогут освободить захваченные ресурсы. Поэтому ни один из потоков не сможет продолжать выполнение, что означает наличие взаимоблокировки.

В целом, использование нескольких потоков может значительно повысить производительность вашего приложения, но требует особого внимания к синхронизации доступа к разделяемым данным и обработке исключений.

Задание на лабораторную работу

Смоделировать ситуацию взаимоблокировки согласно варианту (запрограммировать и показать данную ситуацию).

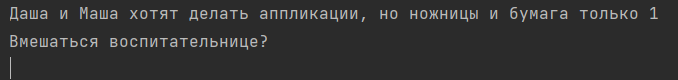
**Вариант 5.** Две девочки Маша и Даша в детском саду делают аппликацию. Для работы каждой нужны ножницы и цветная бумага. Предположим Маша взяла ножницы (поток Маша вошла в монитор объекта ножницы), а Даша бумагу (поток Даша вошла в монитор объекта бумага). Каждая из них ждет другой предмет и не хочет делиться тем, что взяла. Они не могут продолжить свою работу и будут ждать вечно (пока воспитательница не поможет им).

Построить граф распределения ресурсов и пояснить его. Применить алгоритм для обнаружения взаимоблокировки при использовании одного ресурса каждого типа.

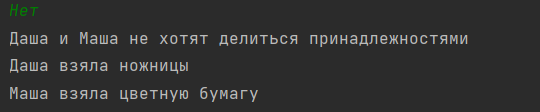
Произвести выход из взаимоблокировки и обосновать метод выхода.

Дать разъяснение методам обработки взаимоблокировок на выбранном языке программирования.

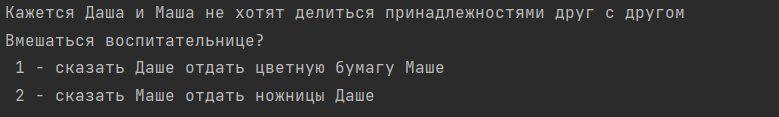
Пример работы программы:

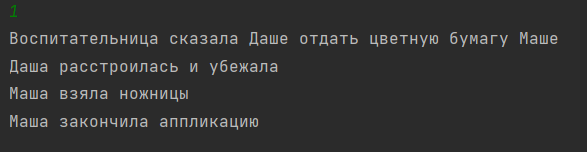


Если ввести «Нет», потоки застрянут во взаимной блокировке



Появится предложение о вмешательстве (прерывании потока):

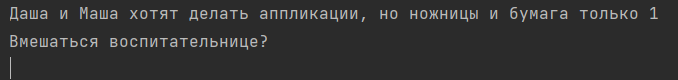




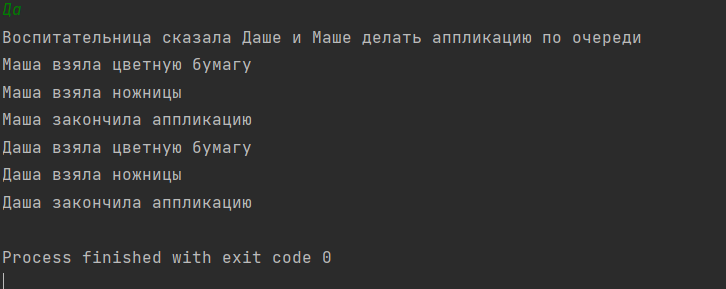
Был прерван поток Даши. Поток Маши завершил свое выполнение

Выход из взаимоблокировки был осуществлен методом прерывания потока.

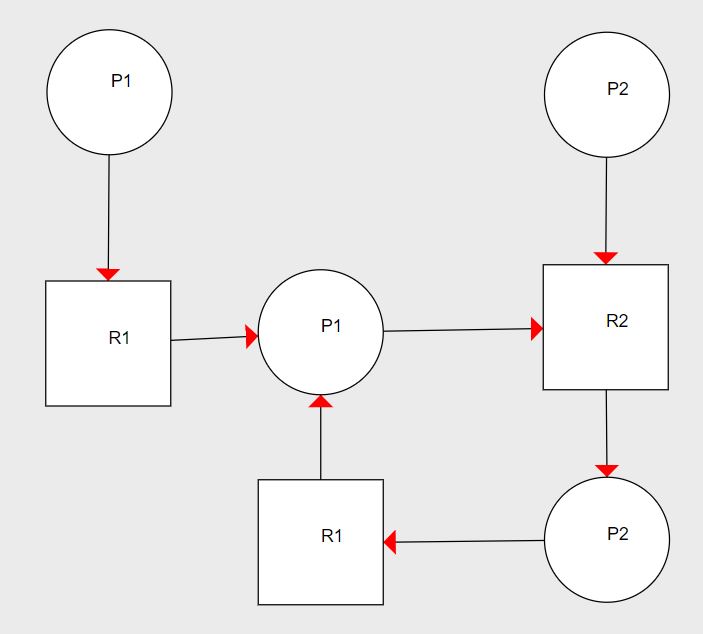
Если на вопрос



Ответить «Да», то будет запущено выполнение с предотвращенной взаимоблокировкой:



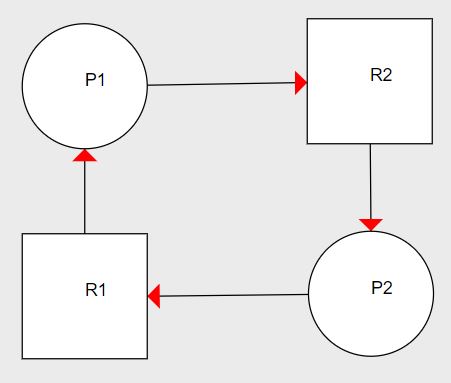
граф распределения ресурсов:



Здесь P1 и P2 – потоки Маша и Даша соответственно,

R1 и R2 – ресурсы – Цветная бумага и ножницы.

Маша запрашивает цветную бумагу, Даша запрашивает ножницы. У них обеих получается получить доступ к ресурсам, и они захватывают их. Далее Маша пытается захватить ножницы, но они уже захвачены Дашей, в то время как Даша пытается захватить цветную бумагу, которая захвачена Машей. Образуется цикл, в котором Маша, не освобождая цветной бумаги, запрашивает ножницы, которые Даша отдаст тогда, когда Маша освободит цветную бумагу:



Методы обработки взаимоблокировок на java:

1. Метод tryLock() класса Lock пытается получить блокировку на объекте, но если блокировка не может быть получена в данный момент, то метод возвращает false. Этот метод может использоваться вместо блокировки методом lock(), чтобы избежать взаимоблокировок.
2. Метод lockInterruptibly() класса Lock позволяет получить блокировку на объекте, но может быть прерван, если другой поток вызывает метод interrupt() на текущем потоке.
3. Методы await() и signal() класса Condition позволяют потокам работать с блокировками в более гибком режиме. Метод await() позволяет потоку освободить блокировку и перейти в ожидание на условии, а метод signal() позволяет потоку уведомить другой поток, который ожидает на этом условии, что оно было выполнено.
4. Методы tryLock() и lockInterruptibly() класса ReentrantReadWriteLock позволяют получить блокировку на объекте в зависимости от типа блокировки (чтение или запись). Таким образом, если несколько потоков пытаются получить блокировку на одном объекте для чтения, то они могут это сделать одновременно.
5. Методы tryLock(long time, TimeUnit unit) и lockInterruptibly(long time, TimeUnit unit) класса Lock позволяют получить блокировку на объекте в течение определенного времени, указанного в параметрах метода. Если блокировка не может быть получена в течение этого времени, то метод возвращает false.

Листинг программы:

import java.util.Scanner;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

class DeadlockExample {

private final Lock paper = new ReentrantLock();//Ресурс цветная бумана

private final Lock scissors = new ReentrantLock(); //Ресурс ножницы

public void mashaDo(){

try {

paper.lockInterruptibly();

System.out.println("Маша взяла цветную бумагу");

Thread.sleep(1000);

} catch (InterruptedException ignored) {}

try {

scissors.lockInterruptibly();

System.out.println("Маша взяла ножницы");

Thread.sleep(1000);

System.out.println("Маша закончила аппликацию");

scissors.unlock();

} catch (InterruptedException ignored) {}

paper.unlock();

}

public void dashaDo(){

try {

scissors.lockInterruptibly();

System.out.println("Даша взяла ножницы");

Thread.sleep(1000);

} catch (InterruptedException ignored) {}

try {

paper.lockInterruptibly();

System.out.println("Даша взяла цветную бумагу");

Thread.sleep(1000);

System.out.println("Даша закончила аппликацию");

paper.unlock();

} catch (InterruptedException ignored) {}

scissors.unlock();

}

public void mashaDoSync(){

try {

paper.lock();

System.out.println("Маша взяла цветную бумагу");

Thread.sleep(1000);

scissors.lock();

System.out.println("Маша взяла ножницы");

Thread.sleep(1000);

System.out.println("Маша закончила аппликацию");

scissors.unlock();

paper.unlock();

} catch (InterruptedException ignored) {}

}

public void dashaDoSync(){

try {

paper.lock();

System.out.println("Даша взяла цветную бумагу");

Thread.sleep(1000);

scissors.lock();

System.out.println("Даша взяла ножницы");

Thread.sleep(1000);

System.out.println("Даша закончила аппликацию");

scissors.unlock();

paper.unlock();

} catch (InterruptedException ignored) {}

}

}

public class Main{

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

System.out.println("Даша и Маша хотят делать аппликации, но ножницы и бумага только 1");

System.out.println("Вмешаться воспитательнице?");

String choice = in.nextLine();

switch (choice) {

case "Да" -> {

System.out.println("Воспитательница сказала Даше и Маше делать аппликацию по очереди");

controlledWork();

}

case "Нет" -> {

System.out.println("Даша и Маша не хотят делиться принадлежностями");

uncontrolledWork();

}

default -> System.out.println("Маша и Даша будут вечно ждать друг друга");

}

}

static void controlledWork(){

DeadlockExample deadlock = new DeadlockExample();

Thread mashaWork = new Thread(deadlock::mashaDoSync);

Thread dashaWork = new Thread(deadlock::dashaDoSync);

mashaWork.start();

dashaWork.start();

}

static void uncontrolledWork(){

DeadlockExample deadlock = new DeadlockExample();

Thread mashaWork = new Thread(deadlock::mashaDo);

Thread dashaWork = new Thread(deadlock::dashaDo);

mashaWork.start();

dashaWork.start();

try {

Thread.sleep(1000);

} catch (InterruptedException ignored) {}

System.out.println("Кажется Даша и Маша не хотят делиться принадлежностями друг с другом \nВмешаться воспитательнице? \n 1 - сказать Даше отдать цветную бумагу Маше \n 2 - сказать Маше отдать ножницы Даше");

Scanner in = new Scanner(System.in);

int choice = in.nextInt();

switch (choice) {

case 1 -> {

System.out.println("Воспитательница сказала Даше отдать цветную бумагу Маше");

dashaWork.interrupt();

System.out.println("Даша расстроилась и убежала");

}

case 2 -> {

System.out.println("Воспитательница сказала Маше отдать ножницы Даше");

mashaWork.interrupt();

System.out.println("Маша расстроилась и убежала");

}

default -> System.out.println("Маша и Даша будут вечно ждать друг друга");

}

}

}