Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Саратовский государственный технический университет

имени Гагарина Ю.А.»

Институт электронной техники и приборостроения

Кафедра Информационная безопасность автоматизированных систем

Направление 10.05.03 Информационная безопасность автоматизированных систем

**Практическая работа №3**

по дисциплине «Параллельные системы и их программирование»

|  |  |
| --- | --- |
|  | Выполнил: студент 4 курса  учебной группы с-ИБС42  очной формы обучения  Солодилов В.В.  Проверил: профессор. каф. ИБС  Кондратов Д.В. |

Саратов 2022

**Задание**

1. Реализуйте взаимодействие потоков-читателей и потоков-писателей с общим буфером без каких-либо средств синхронизации. Проиллюстрируйте проблему совместного доступа. Почему возникает проблема доступа?
2. Реализуйте доступ "читателей" и "писателей" к буферу с применением следующих средств синхронизации:
   * блокировки (lock);
   * сигнальные сообщения (ManualResetEvent, AutoResetEvent, ManualResetEventSlim);
   * семафоры (Semaphore, SemaphoreSlim).
   * атомарные операторы (Interlocked)
3. Исследуйте производительность средств синхронизации при разном числе сообщений, разном объеме сообщений, разном числе потоков.
4. Сделайте выводы об эффективности применения средств синхронизации.

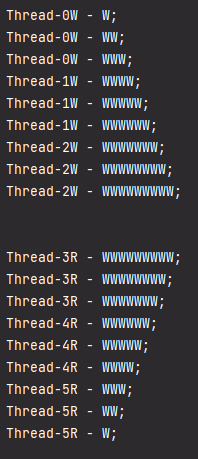
**Вопросы**

1. Почему проблема гонки данных проявляется не при каждом прогоне?
2. Какие факторы увеличивают вероятность проявления проблемы гонки данных?
3. Возможно ли в данной задаче при отсутствии средств синхронизации возникновение исключения и аварийное завершение программы?
4. Можно ли в данной задаче использовать атомарные операторы для обеспечения согласованности доступа? Необходимы ли при этом дополнительные средства синхронизации?
5. Можно ли в данной задаче использовать потокобезопасные коллекции для обеспечения согласованного доступа?
6. Какие средства синхронизации обеспечивают наилучшее быстродействие в данной задаче? Объясните с чем это связано.

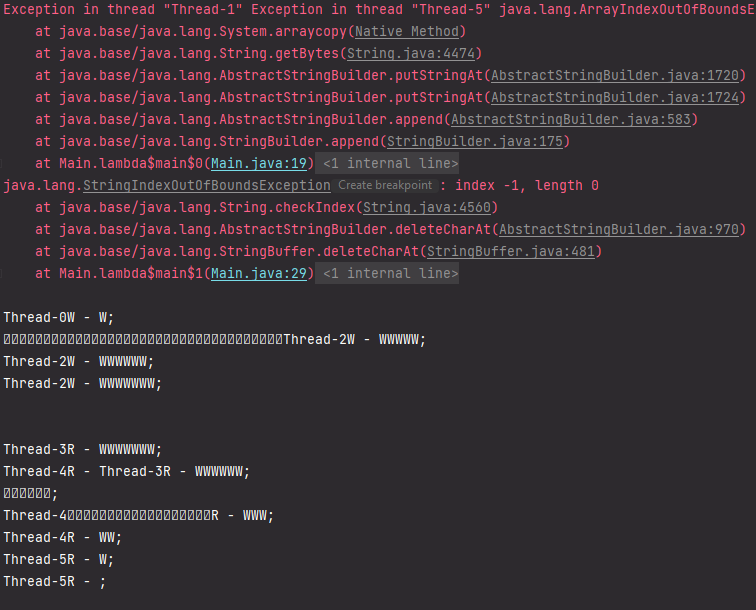
*Задание 1.* На языке программирования Java произведена попытка реализации взаимодействия потоков-читателей и потоков писателей с общим буфером без средств синхронизации. При выполнении задания была выявлена проблема совместного доступа к буферу, которая заключается в возникновении «гонки» потоков. Проявляется данная проблема не всегда, однако её проявление не позволяет безошибочно выполнить задание без средств синхронизации.

Во время работы программы возможны следующие результаты:

1. Программа завершила работу без ошибок:



1. В результате возникновения «гонки» потоков приложение выбросила исключение, завершив работу:



*Задание 2.* Для реализации совместного доступа потоков к буферу памяти были использованы средства синхронизации, такие как блокировки, семафоры и атомарные операторы. По итогу выполнения данного задания можно сказать, что существенно заметна эффективность применения средств синхронизации для обеспечения общего доступа к буферу.

*Задание 3.* Проведём исследование производительности средств синхронизации, использованных ранее. Результаты представим в виде таблицы, где R – число читателей, W – число писателей, M – размер сообщения:

|  |  |  |  |
| --- | --- | --- | --- |
|  | R = 3; W = 3; M = 3 | R = 10; W = 10; M = 15 | R = 20; W = 15; M = 15 |
| Lock | 3 мс | 3 мс | 4 мс |
| Semaphore | 2 мс | 3 мс | 4 мс |

*Задание 4.* По результатам, представленным в таблице, можно сделать вывод, что среднее время работы программы при разных значениях числа читателей, писателей и длины сообщения при малых значениях практически не отличаются, что позволяет в равной мере использовать данные средства синхронизации без особой потери производительности.

**Ответы на вопросы**

1. Появление проблемы «гонки» зависит от того, сколько данных поток успел «захватить» до возникновения данной ошибки. Если при получении данных одним потоком другой поток получает другие данные, то такой ошибки не возникает. Однако такое происходит далеко не при каждом запуске; в определенный момент может возникнуть ситуация, когда оба потока обращаются к одним и тем же данным, в результате чего проявляется проблема гонки.
2. Главным фактором проявления данной проблемы является отсутствие или неправильная настройка средств синхронизации данных.
3. В данной задаче возможно возникновение как исключений, так и аварийного завершения работы, что было продемонстрировано на первом рисунке при отсутствии средств синхронизации
4. В данной задаче возможно использование атомарных операторов, однако для их правильной работы необходима реализация дополнительных средств синхронизации, таких как Lock.
5. Потокобезопасные коллекции могут выступать одним из средств синхронизации, включая в себя непосредственно алгоритм синхронизации и переменные для хранения данных.
6. В данной задаче практически отсутствует влияние средства синхронизации на быстродействие программы, что позволяет использовать любое из вышеприведенных в зависимости от ситуации. Во многом это связано с небольшой выборкой исходных данных, однако при их увеличении разница остаётся на уровне погрешности.

**Листинг программы**

**//Файл Main.java**

import java.util.ArrayList;

public class Main {

public static void main(String[] args) throws InterruptedException {

final int writingsCount = 3;

final int writersCount = 3;

final int readersCount = 3;

StringBuffer buf = new StringBuffer();

StringBuilder stringBuilderWriters = new StringBuilder();

StringBuilder stringBuilderReaders = new StringBuilder();

ArrayList<Thread> threadsWriters = new ArrayList<>();

for (int i = 0; i < writersCount; i++) {

threadsWriters.add(new Thread(() -> {

for (int j = 0; j < writingsCount; j++) {

buf.append("W");

stringBuilderWriters.append(Thread.currentThread().getName()).append("W - ").append(buf).append(";\n");

}

}));

}

ArrayList<Thread> threadsReaders = new ArrayList<>();

for (int i = 0; i < readersCount; i++) {

threadsReaders.add(new Thread(() -> {

for (int j = 0; j < readersCount; j++) {

stringBuilderReaders.append(Thread.currentThread().getName()).append("R - ").append(buf).append(";\n");

buf.deleteCharAt(buf.length() - 1);

}

}));

}

for (Thread thread : threadsWriters) thread.start();

for (Thread thread : threadsReaders) thread.start();

System.out.println("\n" + stringBuilderWriters);

System.out.println("\n" + stringBuilderReaders);

}

}

**//Файл Buffer.java**

package sync;

import java.util.concurrent.TimeUnit;

import java.util.concurrent.locks.Lock;

import java.util.concurrent.locks.ReentrantLock;

public class Buffer<T> {

private T buffer = null;

private final Lock RWLock = new ReentrantLock();

public boolean writeInBuffer(T newObject) {

boolean writingSuccess = false;

try {

if (RWLock.tryLock(10, TimeUnit.MILLISECONDS)) {

if (buffer != null) throw new Exception("Buffer is full");

else {

buffer = newObject;

writingSuccess = true;

}

}

} catch (Exception e) {

//e.printStackTrace();

} finally {

RWLock.unlock();

}

return writingSuccess;

}

public T readFromBuffer() {

T bufferObject = null;

try {

if (RWLock.tryLock(10, TimeUnit.SECONDS)) {

if (buffer == null) throw new Exception("Buffer is empty");

else {

bufferObject = buffer;

buffer = null;

}

}

} catch (Exception e) {

//e.printStackTrace();

} finally {

RWLock.unlock();

}

return bufferObject;

}

}

**//Файл Lock.java**

package sync;

import java.util.ArrayList;

import java.util.concurrent.atomic.AtomicInteger;

public class Lock extends ReadWrite {

private static final AtomicInteger finishOfWriting = new AtomicInteger(0);

public static void main(String[] args) throws InterruptedException {

long timeMillis = System.currentTimeMillis();

final Buffer<Integer> buffer = new Buffer<>();

final int writingCount = 100;

final int writersCount = 100;

final int readersCount = 100;

ArrayList<Thread> threadsWriters = new ArrayList<>();

for (int i = 0; i < writersCount; i++) {

threadsWriters.add(new Thread(() -> {

for (int j = 0; j < writingCount; ) {

if (buffer.writeInBuffer(j)) {

write(writingCount, j);

j++;

}

}

finishOfWriting.incrementAndGet();

}));

threadsWriters.get(i).join();

}

ArrayList<Thread> threadsReaders = new ArrayList<>();

for (int i = 0; i < readersCount; i++) {

threadsReaders.add(new Thread(() -> {

while (finishOfWriting.get() < writersCount - 1) {

Integer current = buffer.readFromBuffer();

if (current != null) {

read(writingCount, current);

}

}

}));

threadsReaders.get(i).join();

}

for (Thread thread : threadsWriters) thread.start();

for (Thread thread : threadsReaders) thread.start();

System.out.println("\nProgram completed in " + ((System.currentTimeMillis() - timeMillis)) + " milliseconds.");

}

}

**//Файл Semaphore.java**

package *sync*;  
  
import *java.util.ArrayList*;  
import *java.util.concurrent.atomic.AtomicInteger*;  
  
public class *Semaphore* extends *ReadWrite* {  
 private static final *AtomicInteger* endWriting = new AtomicInteger(0);  
 private static *Integer buffer* = null;  
 private static final *java.util.concurrent.Semaphore* semaphoreRead = new *java.util.concurrent.*Semaphore(1, true);  
 private static final *java.util.concurrent.Semaphore* semaphoreWrite = new *java.util.concurrent.*Semaphore(1 , true);  
  
 public static void main(*String*[] *args*) {  
  
 long millis = *System*.*currentTimeMillis*();  
 final int writingCount = 100;  
 final int writersCount = 100;  
 final int readersCount = 100;  
  
 *ArrayList*<*Thread*> threadsWriters = new ArrayList<>();  
 for (int i = 0; i < writersCount; i++) {  
 threadsWriters.add(new Thread(() -> {  
 for (int j = 0; j < writingCount; j++) {  
 try {  
 semaphoreWrite.acquire();  
 if (*buffer* == null) {  
 *buffer* = j;  
 *write*(writingCount, j);  
 }  
 } catch (*InterruptedException e*) {  
 *e*.printStackTrace();  
 } finally {  
 semaphoreWrite.release();  
 }  
 }  
 endWriting.incrementAndGet();  
 }));  
 }  
  
 *ArrayList*<*Thread*> threadsReaders = new ArrayList<>();  
 for (int i = 0; i < readersCount; i++) {  
 threadsReaders.add(new Thread(() -> {  
 while (endWriting.get() < writersCount - 1) {  
 try {  
 semaphoreRead.acquire();  
 if (*buffer* != null){  
 int current = *buffer*;  
 *buffer* = null;  
 *read*(writingCount, current);  
 }  
 } catch (*InterruptedException e*) {  
 *e*.printStackTrace();  
 } finally {  
 semaphoreRead.release();  
 }  
 }  
 }));  
 }  
  
 for (*Thread* thread : threadsWriters) {  
 thread.start();  
 }  
 for (*Thread* thread : threadsReaders) {  
 thread.start();  
 }  
  
 *System*.out.println("\nProgram continued " + ((*System*.*currentTimeMillis*() - millis)) + " milliseconds.");  
 }  
}

**//Файл Operations.java**

package sync;

public class Operations {

public static void read(int nWritings, int current) {

StringBuilder tmp = new StringBuilder();

tmp.append("\t").append(Thread.currentThread().getName()).append("R - ");

for (int k = 0; k < nWritings; k++) {

tmp.append(k > current ? "\_" : "R");

}

System.out.println(tmp);

}

public static void write(int nWritings, int current) {

StringBuilder tmp = new StringBuilder();

tmp.append(Thread.currentThread().getName()).append("W - ");

for (int k = 0; k < nWritings; k++) {

tmp.append(k > current ? "\_" : "W");

}

System.out.println(tmp);

}

}