Реализация механизма внесения контролируемых изменений в порядок многопоточного исполнения для библиотеки LinCheck

Автор: Пыхтин А.В.
Научный руководитель: Цителов Д.И.

Проблемы параллельных алгоритмов

- Состояние гонки
- Взаимная блокировка
- Трудности коммуникации

LinCheck

LinCheck - библиотека, проверяющая многопоточные структуры на линеаризуемость.

Линеаризуемость– свойство программы, при котором результат любого параллельного выполнения операций эквивалентен некоторому последовательному

Постановка задачи

Необходимо реализовать механизм внесения контролируемых изменений в порядок многопоточного исполнения библиотеки LinCheck, что позволит улучшить распределение результатов параллельного исполнения и повысит вероятность обнаружения ошибки в многопоточных структурах данных.

Ход работы

- **Изменение API**
- Генерация классов
- Контролируемые задержки

Изменения API

- Взаимодействие с тестируемой структурой
- Написание генераторов

```
1  @CTest(iter = 300, actorsPerThread = {"1:3", "1:3", "1:3"})
2  @Param(name = "key", clazz = FloatGenerator.class, opt = {"0", "10", "0.1"})
3  public class SetAndGetTest {
4      public SetAndGet setAndGet;
5      @Reset
6      public void reload() {
7          setAndGet = new SetAndGet();
8      }
9      @Operation(params = {"key"})
10     public float setAndGet(float key) throws Exception{
11         return setAndGet.setAndGet(key);
12     }
13     @Test
14     public void test() throws Exception {
15         Checker checker = new Checker();
16         assertFalse(checker.checkAnnotated(new SetAndGetTest()));
17     }
18 }
```

Ход работы

- Изменение API
- **Генерация классов**
- Контролируемые задержки

Генерация классов

- Взаимодействие с любыми типами данных
- Увеличение количества методов в потоке

```
1  public class Generated10 extends Generated {
2      public Object testClass;
3      private Barier barier;
4      public Generated10(Object testClass, Barier barier) {
5          this.testClass = testClass;
6          this.barier = barier;
7      }
8      public void process(Result[] res, Object[][] args, int[] waits, int[] offset) {
9          int a = (int)args[0][0];
10         for (int i = 0; i < offset[0]; i++) {
11             barier.waitOtherTread();
12         }
13         try{
14             MyRandom.busyWait(waits[0]);
15             res[0].setValue(testClass.CallMethod(a));
16         }catch (Exception e) {
17             res[0].setException(e);
18         }
19 ...
20       barier.waitOtherTread();
21     }
22 }
```

Ход работы

- Изменение API
- Генерация классов
- **Контролируемые задержки**

Обзор

Добавление временных задержек

- Случайные задержки
- Задержки, основанные на подсчете времени выполнения методов
- Задержки, основанные на подсчете времени вызова метода

Добавление синхронизатора

- Параллельное выполнение в определенных местах
- Последовательное выполнение части команд

Результаты

Конфигурация

№ Потока	Операции в потоке		
1	Put(0)	Get()	
2	Put(9)	Put(9)	Get()
3	Put(5)	Put(7)	

№ Результата	Результат выполнения
1	-10, 0, -10, -10, 9, -10, -10
2	-10, 0, -10, -10, 5, -10, -10
3	-10, 9, -10, -10, 0, -10, -10
4	-10, 5, -10, -10, 0, -10, -10
5	-10, 9, -10, -10, 9, -10, -10
6	-10, 9, -10, -10, 5, -10, -10
7	-10, 5, -10, -10, 9, -10, -10
8	-10, 5, -10, -10, 7, -10, -10
9	-10, 7, -10, -10, 5, -10, -10

Результат

Результаты

Конфигурация

№ Потока	Операции в потоке		
1	Put(7)	Put(2)	
2	Put(2)	Get()	Put(6)
3	Put(0)		

№ Результата	Результат выполнения
1	-10, -10, -10, 7, -10, -10
2	-10, -10, -10, 2, -10, -10
3	-10, -10, -10, 0, -10, -10

Результат

Результаты

Конфигурация

№ Потока	Операции в потоке		
1	Put(5)	Put(2)	
2	Get()		
3	Get()		

№ Результата	Результат выполнения
1	-10, -10, 5, 2
2	-10, -10, 2, 5
3	-10, -10, 5, QueueEmptyException
4	-10, -10, QueueEmptyException, 5
5	-10,-10, QueueEmptyException, QueueEmptyException

Результат