FlatBuffer Program Report

Matt Luckcuck

Department of Computer Science,
University of York, UK

ml881@york.ac.uk

4th February 2016

Contents

1 Introduction

The *FlatBuffer* application is a contrived example to illustrate some of the features of Level 2. FlatBuffer is an SCJ solution to the Readers-Writers Problem, using a one-place buffer. The program is named for its relatively 'flat' program hierarchy – SCJ Level 2 allows much more complex hierarchies than the FlatBuffer. The FlatBuffer is structurally simple but uses two of Level 2's unique features: managed threads and suspension. More interesting examples can be found in [1].

Figure 1 shows an object diagram of the FlatBuffer. The program is controlled by the safelet `Flatbuffer`, which starts the top-level mission sequencer `FlatbufferMissionSequencer`. The mission sequencer starts the `FlatbufferMission`, which starts the two managed threads. One managed thread is a producer, the `Writer`, and the other is the consumer, `Reader`.

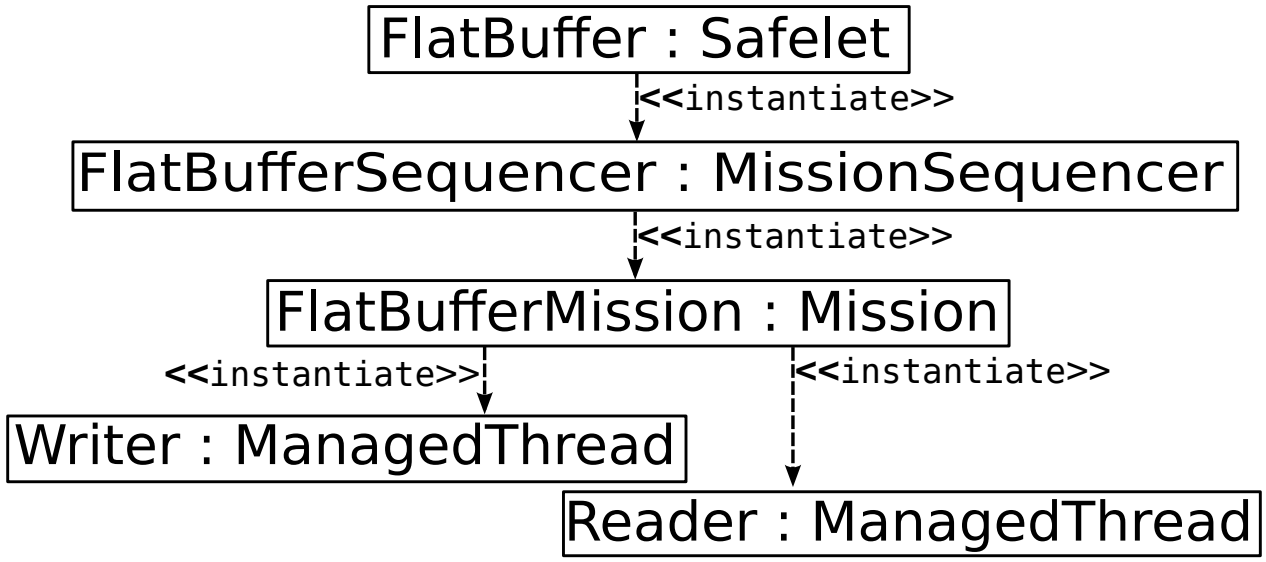

Figure 1: Object Diagram of the Flatbuffer

In the FlatBuffer, the one-place buffer is stored in the `FlatBufferMission`, so that it can be shared by the schedulables `Writer` and `Reader`. It has a `bufferEmpty()` method to check the status of the buffer, a `read()` method to read and reset the buffer, and a `write()` method to update the value of the buffer. The `read()` and `write()` methods are both declared as `synchronized`, which means that the monitor lock of the the `FlatBufferMission` must be obtained by a calling thread before it may execute the method. Further, they use `Object.wait()` and `Object.notify()` to suspend the managed threads when the buffer is full (for the writer) or empty (for the writer) and be woken when they can continue. This ensures that the writer does not write when the buffer is full and the reader does not try to read an empty buffer.

The FlatBuffer is structurally simple, but it makes use of two of the unique features of Level 2. Level 1 applications are not able to use managed threads or make calls to `Object.wait()` or `Object.notify()`. This means Level 2 is ideal for programming applications that use thread-style execution or need to use suspension. Section 2 presents the full script of the FlatBuffer program.

2 Program Script

2.1 FlatBuffer

```
1  public class FlatBuffer implements Safelet<Mission>{
2
3    public Level getLevel(){
4      return Level.LEVEL_2;
5    }
6
7    public MissionSequencer<Mission> getSequencer(){
8
9      StorageParameters storageParameters = new StorageParameters(
10         Const.OVERALL_BACKING_STORE_DEFAULT - 1000000,
11         new long[] { Const.HANDLER_STACK_SIZE },
12         Const.PRIVATE_MEM_DEFAULT, 10000 * 2, Const.MISSION_MEM_DEFAULT);
13
14     return new FlatBufferMissionSequencer(new PriorityParameters(5),
15         storageParameters);
16   }
17
18   public long immortalMemorySize(){
19     return Const.IMMORTAL_MEM_DEFAULT;
20   }
21
22   public void initializeApplication(){
23   }
24 }
```

2.2 FlatBufferMissionSequencer

```
1  public class FlatBufferMissionSequencer extends MissionSequencer<Mission>{
2
3    private boolean returnedMission;
4
5    public FlatBufferMissionSequencer(PriorityParameters priorityParameters,
6        StorageParameters storageParameters){
7      super(priorityParameters, storageParameters);
8      returnedMission = false;
9    }
10
11   protected Mission getNextMission(){
12
13     if (!returnedMission){
14       returnedMission = true;
15       return new FlatBufferMission();
16     }
17     else{
18       return null;
19     }
20   }
21 }
```

2.3 FlatBufferMission

```
1  public class FlatBufferMission extends Mission{
2
3    private volatile int buffer;
4
5    public FlatBufferMission(){
6      Console.println("FlatBufferMission");
7      buffer = 0;
8      Services.setCeiling(this, 20);
9    }
10
11   protected void initialize(){
12     StorageParameters storageParameters = new StorageParameters(150 * 1000,
13         new long[] { Const.HANDLER_STACK_SIZE },
14         Const.PRIVATE_MEM_DEFAULT, Const.IMMORTAL_MEM_DEFAULT,
15         Const.MISSION_MEM_DEFAULT - 100 * 1000);
16
17     new Reader(new PriorityParameters(10), storageParameters, this).register();
18     new Writer(new PriorityParameters(10), storageParameters, this).register();
19   }
20
21   public boolean bufferEmpty(){
22
23     return buffer == 0;
24   }
25
26   public synchronized void write(int update) throws InterruptedException{
27
28     while (!bufferEmpty("Writer")){
29       this.wait();
30     }
31     buffer = update;
32     this.notify();
33   }
34
35   public synchronized int read() throws InterruptedException{
36     while(bufferEmpty("Reader")){
37       this.wait();
38     }
39     int out = buffer;
40     buffer = 0;
41     this.notify();
42     return out;
43   }
44
45   public boolean cleanUp(){
46     return false;
47   }
48
49   public long missionMemorySize(){
50     return 1048576;
51   }
52 }
```

2.4 Writer

```
1  public class Writer extends ManagedThread{
2
3    private final FlatBufferMission fbMission;
4    private int i = 1;
5
6    public Writer(PriorityParameters priority, StorageParameters storage,
7        FlatBufferMission fbMission){
8      super(priority, storage);
9      this.fbMission = fbMission;
10   }
11
12   public void run(){
13     while (!fbMission.terminationPending()){
14       try{
15         fbMission.write(i);
16       }
17       catch (InterruptedException e){
18         e.printStackTrace();
19       }
20
21       i++;
22
23       boolean keepWriting = i >= 5 ;
24       if(!keepWriting){
25         fbMission.requestTermination();
26       }
27     }
28   }
29 }
```

2.5 Reader

```
1  public class Reader extends ManagedThread{
2
3    private final FlatBufferMission fbMission;
4
5    public Reader(PriorityParameters priority, StorageParameters storage,
6        FlatBufferMission fbMission){
7      super(priority, storage);
8      this.fbMission = fbMission;
9    }
10
11   public void run(){
12     while (!fbMission.terminationPending())
13     {
14       int result=999;
15       try{
16         result = fbMission.read();
17       }
18       catch (InterruptedException e){
19         e.printStackTrace();
20       }
21       Console.println("Reader Read " + result + " from Buffer");
22     }
23   }
24 }
```

References

[1] Andy Wellings, Matt Luckcuck, and Ana Cavalcanti. Safety-critical java level 2: motivations, example applications and issues. In *Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems*, JTRES '13, pages 48–57, New York, NY, USA, 9 October 2013. ACM.