Teoria współbieżności

Laboratorium 13

CSP

Albert Gierlach

23.01.2021

1. Ćwiczenie

Zaimplementuj w Javie z użyciem JCSP rozwiązanie problemu producenta i konsumenta z buforem N-elementowym tak, aby każdy element bufora był reprezentowany przez odrębny proces (taki wariant ma praktyczne uzasadnienie w sytuacji, gdy pamięć lokalna procesora wykonującego proces bufora jest na tyle mała, że mieści tylko jedną porcję). Uwzględnij dwie możliwości:

a) kolejność umieszczania wyprodukowanych elementów w buforze oraz kolejność pobierania nie mają znaczenia

b) pobieranie elementów powinno odbywać się w takiej kolejności, w jakiej były umieszczane w buforze

2. Rozwiązanie

Na podstawie podanego pseudokodu zaimplementowałem oba warianty z wykorzystaniem biblioteki JCSP.

2.1. Wariant bez uwzględnienia kolejności

```
public class Main {
    static final int buffersNum = 10;
    static final int itemsNum = 10000;

    public static void main(String[] args) {
        var channelIntFactory = new StandardChannelIntFactory();
        var prodChannel = channelIntFactory.createOne2One(buffersNum);
        var consChannel = channelIntFactory.createOne2One(buffersNum);
        var bufferChannel = channelIntFactory.createOne2One(buffersNum);

        var procList = new CSProcess[buffersNum + 2];
        procList[0] = new Producer(prodChannel, bufferChannel, itemsNum);
        procList[1] = new Consumer(consChannel, itemsNum);

        IntStream.range(0, buffersNum).forEach(i -> {
            procList[i + 2] =
                    new Buffer(
                            prodChannel[i],
                            consChannel[i],
                            bufferChannel[i]
                    );
```

```
        });

        new Parallel(procList).run();
    }
}

class Producer implements CSProcess {
    private final One2OneChannelInt[] out;
    private final One2OneChannelInt[] jeszcze;
    private final int n;

    public Producer(One2OneChannelInt[] out, One2OneChannelInt[] jeszcze, int n) {
        this.out = out;
        this.jeszcze = jeszcze;
        this.n = n;
    }

    public void run() {
        var guards = new Guard[jeszcze.length];
        for (int i = 0; i < out.length; i++) {
            guards[i] = jeszcze[i].in();
        }

        var alt = new Alternative(guards);
        for (int i = 0; i < n; i++) {
            var index = alt.select();
            jeszcze[index].in().read();

            var item = (int) (Math.random() * 100) + 1;
            out[index].out().write(item);
        }
    }
}

class Consumer implements CSProcess {
    private final One2OneChannelInt[] in;
    private final int n;

    public Consumer(final One2OneChannelInt[] in, int n) {
        this.in = in;
        this.n = n;
    }
```

```
    public void run() {
        var start = System.currentTimeMillis();
        var guards = new Guard[in.length];
        for (int i = 0; i < in.length; i++)
            guards[i] = in[i].in();

        var alt = new Alternative(guards);
        for (int i = 0; i < n; i++) {
            int index = alt.select();
            int item = in[index].in().read();
        }

        var end = System.currentTimeMillis();
        System.out.println((end - start) + "ms");
        System.exit(0);
    }
}

class Buffer implements CSProcess {
    private final One2OneChannelInt in;
    private final One2OneChannelInt out;
    private final One2OneChannelInt jeszcze;

    public Buffer(One2OneChannelInt in,
                  One2OneChannelInt out,
                  One2OneChannelInt jeszcze) {
        this.out = out;
        this.in = in;
        this.jeszcze = jeszcze;
    }

    public void run() {
        while (true) {
            jeszcze.out().write(0);
            out.out().write(in.in().read());
        }
    }
}
```

Wykonanie programu dało następujące wyniki:

```
178ms
```

2.2. Wariant z uwzględnieniem kolejności

```
public class Main {
    static final int buffersNum = 10;
    static final int itemsNum = 10000;

    public static void main(String[] args) {
        var channelIntFactory = new StandardChannelIntFactory();
        var channels = channelIntFactory.createOne2One(buffersNum + 1);

        var procList = new CSProcess[buffersNum + 2];
        procList[0] = new Producer(channels[0], itemsNum);
        procList[1] = new Consumer(channels[buffersNum], itemsNum);

        IntStream.range(0, buffersNum).forEach(i -> {
            procList[i + 2] =
                    new Buffer(
                            channels[i],
                            channels[i + 1]
                    );
        });

        new Parallel(procList).run();
    }
}

class Producer implements CSProcess {
    private final One2OneChannelInt out;
    private final int n;

    public Producer(One2OneChannelInt out, int n) {
        this.out = out;
        this.n = n;
    }

    public void run() {
        for (int i = 0; i < n; i++) {
            var item = (int) (Math.random() * 100) + 1;
            out.out().write(item);
        }
    }
}
```

```
class Consumer implements CSProcess {
    private final One2OneChannelInt in;
    private final int n;

    public Consumer(final One2OneChannelInt in, int n) {
        this.in = in;
        this.n = n;
    }

    public void run() {
        var start = System.currentTimeMillis();
        for (int i = 0; i < n; i++) {
            int item = in.in().read();
        }

        var end = System.currentTimeMillis();
        System.out.println((end - start) + "ms");
        System.exit(0);
    }
}

class Buffer implements CSProcess {
    private final One2OneChannelInt in;
    private final One2OneChannelInt out;

    public Buffer(One2OneChannelInt in,
                  One2OneChannelInt out) {
        this.out = out;
        this.in = in;
    }

    public void run() {
        while (true) {
            out.out().write(in.in().read());
        }
    }
}
```

Wykonanie programu dało następujące wyniki:

```
211ms
```

3. Wnioski

Biblioteka JCSP dostarcza użytkownikowi przyjaznego interfejsu do modelowania problemów z dziedziny współbieżności. Za jej pomocą udało się zaimplementować problem producenta i konsumera z rozproszonym buforem. Porównując czasy wykonania powyższych programów można wyciągnąć wniosek, że wersja uwzględniająca kolejność pobierania elementów jest nieco wolniejsza od wersji, która tej kolejności nie zachowuje. Dzieje się tak dlatego, że dbanie o kolejność zajmuje dodatkowy czas procesora, ale mimo tego różnice nie są bardzo duże względem siebie.

4. Bibliografia

- https://www.cs.kent.ac.uk/projects/ofa/jcsp/
- https://www.ibm.com/developerworks/java/library/j-csp2/
- https://arild.github.io/csp-presentation/#1
- https://www.cs.cmu.edu/~crary/819-f09/Hoare78.pdf