Laboratorium 13

Łukasz Wala
AGH, Wydział Informatyki, Elektroniki i Telekomunikacji
Teoria Współbieżności 2022/23

Kraków, 10 stycznia 2023

1 Zadania

Zaimplementuj w Javie z użyciem JCSP rozwiązanie problemu pro-ducenta i konsumenta z buforem N-elementowym tak, aby każdy ele-ment bufora był reprezentowany przez odrębny proces (taki wariantma praktyczne uzasadnienie w sytuacji, gdy pamięć lokalna proceso-ra wykonującego proces bufora jest na tyle mała, że mieści tylko jednąporcję). Uwzględnij dwie możliwości:

a) kolejność umieszczania wyprodukowanych elementów w buforzeoraz kolejność pobierania nie mają znaczenia.

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

2 Wariant a - bez uwzględnienia kolejności

W tym wariancie każdy element bufora ma referencję do trzech kanałów (producenta, konsumenta i kanału sygnalizującego, czy dana komórka bufora jest dostępna). Każda komórka bufora będzie w nieskończoność sygnalizować, że jest dostępna. Wówczas jeden z producentów będzie do niej zapisywał dane, a jeden z konsumentów je odczytywał.

```
class Producer implements CSProcess {
    private One2OneChannelInt[] out;
    private One2OneChannelInt[] yet;
    private int n;

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

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

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

            var item = n;
            out[idx].out().write(item);
        }
    }
}

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

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

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

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

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

```
class Buffer implements CSProcess {
    private One2OneChannelInt in;
    private One2OneChannelInt out;
    private One2OneChannelInt yet;

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

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

class Main {
    public static void main(String[] args) {
        int buffers = 100;
        int items = 10000;
        var channelIntFactory = new StandardChannelIntFactory();
        var prodChannel = channelIntFactory.createOne2One(buffers);
        var consChannel = channelIntFactory.createOne2One(buffers);
        var bufferChannel = channelIntFactory.createOne2One(buffers);

        var procs = new CSProcess[buffers + 2];
        procs[0] = new Producer(prodChannel, bufferChannel, items);
        procs[1] = new Consumer(consChannel, items);

        for (int i=0; i<buffers; ++i) {
            procs[i+2] = new Buffer(prodChannel[i], consChannel[i], bufferChannel[i]);
        }

        new Parallel(procs).run();
    }
}
```

Czas wykonania programu wynosi 175 ms.

3 Wariant b - z uwzględnieniem kolejności

W wariancie z uwzględnieniem kolejności dane muszą być przetwarzane po kolei. Każdy elemnt w buforze będzie przechowywał referencję do swojego poprzed-

nika. Dane będą przekazywane z komórki n do $n+1$ dla każdej komórki, aż w końcu zostaną przekazane do konsumenta, co zapewnia odpowiednią kolejność.

```
class Producer implements CSProcess {
    private One2OneChannel out;
    private 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 = n;
            out.out().write(item);
        }
    }
}

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

    public Consumer(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("Time: " + (end - start) + " ms");
        System.exit(0);
    }
}

class Buffer implements CSProcess {
    private One2OneChannelInt in;
    private 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());
        }
    }
}

class Main {
    public static void main(String[] args) {
        int buffers = 100;
        int items = 10000;
        var channelIntFactory = new StandardChannelIntFactory();
        var channels = channelIntFactory.createOne2One(buffers + 1);

        var procs = new CSProcess[buffers + 2];
        procs[0] = new Producer(channels[0], items);
        procs[1] = new Consumer(channels[buffers], items);

        for (int i=0; i<buffers; ++i) {
            procs[i+2] = new Buffer(channels[i], channels[i + 1]);
        }

        new Parallel(procs).run();
    }
}
```

Czas wykonania programu wynosi 208 ms.

4 Wnioski

Biblioteka JCSP pozwala na wygodne modelowanie problemów za pomocą konstrukcji CSP za pomocą interfejsu w Javie. W zaimplementowanym za jej pomocą poblemie producenta i konsumenta czasy wykonania były nieznacząco wyższe w wariancie uwzględniającym kolejność, czego można było się spodziewać patrząc na nieco bardziej skoplikowaną metodę.

5 Bibliografia

1. JCSP web site: https://www.cs.kent.ac.uk/projects/ofa/jcsp/