

1. Zagłódzenie

a. Zmienna boolean

Jeśli używamy zmiennej boolean do zagłódzenia nie dochodzi. Proces A czeka w momencie kiedy nie ma wystarczających zasobów by mógł wykonać swoje zadanie, a reszta procesów (tego samego typu co proces A) czeka aż proces A zakończy wykonywanie zadania i dopiero wtedy następny proces jest dopuszczany do zasobów. Nie ma więc możliwości by proces czekał w nieskończoność na zasoby ponieważ kolejność dostępu do zasobów nie zależy od tego jak wiele ich potrzeba.

```
Run: zadboolean
610 results
(producer3 (34184)) produced(1): [9] -> size: 6
(producer3 (34185)) produced(1): [7] -> size: 7
(producer3 (34186)) produced(1): [3] -> size: 8
(producer3 (34187)) produced(1): [1] -> size: 9
(producer3 (34188)) produced(1): [8] -> size: 10
(producer3 (34189)) produced(1): [2] -> size: 11
(producer3 (34190)) produced(1): [7] -> size: 12
consumer3 is waiting (not first)
producer0 is waiting (not enough space (size:12, max size:20, to insert:10))
(consumer0 (34442)) consumed(10): [6, 8, 3, 9, 4, 9, 7, 3, 1, 8] -> size: 2
consumer0 is waiting (not enough items (available: 2, requested: 10))
producer5 is waiting (not first)
producer3 is waiting (not first)
producer7 is waiting (not first)
producer1 is waiting (not first)
producer8 is waiting (not first)
producer9 is waiting (not first)
producer4 is waiting (not first)
producer6 is waiting (not first)
producer2 is waiting (not first)
consumer5 is waiting (not first)
(producer0 (31275)) produced(10): [9, 3, 4, 7, 4, 5, 2, 7, 4, 8] -> size: 12
producer0 is waiting (not enough space (size:12, max size:20, to insert:10))
producer5 is waiting (not first)
(consumer0 (34443)) consumed(10): [2, 7, 9, 3, 4, 7, 4, 5, 2, 7] -> size: 2
consumer0 is waiting (not enough items (available: 2, requested: 10))
consumer6 is waiting (not first)
(producer0 (31276)) produced(10): [3, 5, 9, 4, 4, 9, 1, 3, 5, 8] -> size: 12
producer0 is waiting (not enough space (size:12, max size:20, to insert:10))
producer3 is waiting (not first)
(consumer0 (34444)) consumed(10): [4, 8, 3, 5, 9, 4, 4, 9, 1, 3] -> size: 2
consumer0 is waiting (not enough items (available: 2, requested: 10))
```

```
Run: zadboolean
643 results
producer1 is waiting (not first)
(consumer0 (34446)) consumed(10): [7, 3, 7, 3, 3, 6, 9, 9, 9, 5] -> size: 2
consumer0 is waiting (not enough items (available: 2, requested: 10))
consumer2 is waiting (not first)
(producer0 (31279)) produced(10): [4, 5, 8, 7, 8, 1, 5, 2, 6, 7] -> size: 12
producer0 is waiting (not enough space (size:12, max size:20, to insert:10))
producer8 is waiting (not first)
(consumer0 (34447)) consumed(10): [7, 6, 4, 5, 8, 7, 8, 1, 5, 2] -> size: 2
consumer0 is waiting (not enough items (available: 2, requested: 10))
consumer8 is waiting (not first)
(producer0 (31280)) produced(10): [7, 2, 6, 4, 4, 9, 6, 4, 9, 5] -> size: 12
producer0 is waiting (not enough space (size:12, max size:20, to insert:10))
producer9 is waiting (not first)
(consumer0 (34448)) consumed(10): [6, 7, 7, 2, 6, 4, 4, 9, 6, 4] -> size: 2
(consumer1 (25872)) consumed(1): [9] -> size: 1
(consumer1 (25873)) consumed(1): [5] -> size: 0
consumer1 is waiting (not enough items (available: 0, requested: 1))
consumer8 is waiting (not first)
(producer0 (31281)) produced(10): [2, 1, 5, 5, 5, 8, 9, 6, 3, 4] -> size: 10
(producer0 (31282)) produced(10): [8, 2, 2, 3, 9, 9, 3, 9, 1, 5] -> size: 20
producer0 is waiting (not enough space (size:20, max size:20, to insert:10))
consumer7 is waiting (not first)
consumer3 is waiting (not first)
producer4 is waiting (not first)
(consumer1 (25874)) consumed(1): [2] -> size: 19
(consumer1 (25875)) consumed(1): [1] -> size: 18
(consumer1 (25876)) consumed(1): [5] -> size: 17
(consumer1 (25877)) consumed(1): [5] -> size: 16
(consumer1 (25878)) consumed(1): [5] -> size: 15
(consumer1 (25879)) consumed(1): [8] -> size: 14
(consumer1 (25880)) consumed(1): [9] -> size: 13
(consumer1 (25881)) consumed(1): [6] -> size: 12
```

b. Metoda hasWaiters()

Jeśli używamy metody `hasWaiters()` dochodzi do zagłódzenia (już po chwili różnica w ilości wykonanych operacji to 1 rząd wielkości). Jest to spowodowane tym, że do kolejki pierwszego procesu jest tak naprawdę wpuszczanych więcej procesów, a więc efekt końcowy jest taki sam jak gdybyśmy użyli tylko po jednej kolejce dla każdego typu procesów. Skutkuje to tym, że za każdym razem gdy nie ma dostępnych wystarczających zasobów dla procesu A, trafia do kolejki i inne procesy mogą korzystać z zasobów które były nie wystarczające dla procesu A, co powoduje że przy jego następnym dostępie tych zasobów może znowu brakować.

```
Run: zadhaswaiters × ↺ Aa W * 53 re
↑ Q- consumer0 (
(producer8 (13744)) produced(1): [5] -> size: 13
(producer8 (13745)) produced(1): [8] -> size: 14
(producer8 (13746)) produced(1): [7] -> size: 15
(producer8 (13747)) produced(1): [9] -> size: 16
(producer8 (13748)) produced(1): [6] -> size: 17
(producer8 (13749)) produced(1): [7] -> size: 18
(producer8 (13750)) produced(1): [7] -> size: 19
(producer8 (13751)) produced(1): [8] -> size: 20
producer8 is waiting (not enough space (size:20, max size:20, to insert:1)
(consumer8 (12237)) consumed(1): [1] -> size: 19
(consumer8 (12238)) consumed(1): [4] -> size: 18
(consumer8 (1685)) consumed(10): [3, 5, 8, 7, 4, 1, 6, 4, 4, 8] -> size: 8
consumer8 is waiting (not enough items (available: 8, requested: 10)
(producer1 (15897)) produced(1): [2] -> size: 9
(producer1 (15898)) produced(1): [2] -> size: 10
(producer1 (15899)) produced(1): [3] -> size: 11
(producer6 (12422)) produced(1): [2] -> size: 12
(producer6 (12423)) produced(1): [5] -> size: 13
(producer6 (12424)) produced(1): [7] -> size: 14
(producer6 (12425)) produced(1): [2] -> size: 15
(producer6 (12426)) produced(1): [3] -> size: 16
(producer6 (12427)) produced(1): [5] -> size: 17
(producer6 (12428)) produced(1): [2] -> size: 18
(producer6 (12429)) produced(1): [2] -> size: 19
(producer6 (12430)) produced(1): [9] -> size: 20
producer6 is waiting (not enough space (size:20, max size:20, to insert:1)
(consumer4 (13578)) consumed(1): [5] -> size: 19
(consumer4 (13579)) consumed(1): [8] -> size: 18
(consumer4 (13580)) consumed(1): [7] -> size: 17
(consumer4 (13581)) consumed(1): [9] -> size: 16
(consumer4 (13582)) consumed(1): [6] -> size: 15
(consumer4 (13583)) consumed(1): [7] -> size: 14
(consumer4 (13584)) consumed(1): [8] -> size: 15
```

```
Run: zadhaswaiters ×
↑ Q- producer0 × ↻ Aa W * 259/259 ↑
(consumer9 (14791)) consumed(1): [8] -> size: 0
consumer9 is waiting (not enough items (available: 0, requested: 1))
consumer2 is waiting (not enough items (available: 0, requested: 1))
(producer6 (1481)) produced(10): [2, 3, 2, 5, 8, 7, 7, 1, 6, 4] -> size: 10
(producer6 (1482)) produced(10): [6, 2, 4, 7, 1, 2, 1, 5, 3, 1] -> size: 20
producer6 is waiting (not enough space (size:20, max size:20, to insert:10))
(consumer7 (12745)) consumed(1): [2] -> size: 19
(consumer7 (12746)) consumed(1): [3] -> size: 18
(consumer7 (12747)) consumed(1): [2] -> size: 17
(consumer7 (12748)) consumed(1): [5] -> size: 16
(consumer7 (12749)) consumed(1): [8] -> size: 15
(consumer7 (12750)) consumed(1): [7] -> size: 14
(consumer7 (12751)) consumed(1): [7] -> size: 13
(consumer7 (12752)) consumed(1): [1] -> size: 12
(consumer7 (12753)) consumed(1): [6] -> size: 11
(consumer7 (12754)) consumed(1): [4] -> size: 10
(consumer7 (12755)) consumed(1): [6] -> size: 9
(consumer7 (12756)) consumed(1): [2] -> size: 8
(consumer7 (12757)) consumed(1): [4] -> size: 7
(consumer7 (12758)) consumed(1): [7] -> size: 6
(consumer7 (12759)) consumed(1): [1] -> size: 5
(consumer7 (12760)) consumed(1): [2] -> size: 4
(consumer7 (12761)) consumed(1): [1] -> size: 3
(consumer7 (12762)) consumed(1): [5] -> size: 2
(consumer7 (12763)) consumed(1): [3] -> size: 1
(consumer7 (12764)) consumed(1): [1] -> size: 0
consumer7 is waiting (not enough items (available: 0, requested: 1))
(producer1 (12681)) produced(1): [1] -> size: 1
(producer1 (12682)) produced(1): [5] -> size: 2
(producer1 (12683)) produced(1): [9] -> size: 3
(producer1 (12684)) produced(1): [9] -> size: 4
(producer1 (12685)) produced(1): [9] -> size: 5
```

2. Zakleszczenie

a. Zmienna boolean

Jeśli używamy zmiennej boolean do zakleszczenia nigdy nie dochodzi, nawet przy dużej różnicy między ilością procesów danych typów.

b. Metoda hasWaiters()

Używając metody hasWaiters() do zakleszczenia dochodzi za każdym razem. Czas potrzebny na zakleszczenie jest zależny od ilości producentów i konsumentów, np. dla 5 producentów i 1 konsumenta do zakleszczenia dochodzi praktycznie od razu.