Teoria współbieżności - Zadanie 1

Szymon Twardosz

16 listopada 2023

1 Zagłodzenie - 2 Conditions

Założenia:

- maksymalna pojemność bufora wynosi 10;
- w systemie działają dwa wątki konsumentów i dwa wątki producentów
- wątki konsumentów i producentów mogą domagać się od bufera wartości z przedziału [1, 5]

Układ zdażeń:

1. *Start*
   Lock = {P1, P2, C1, C2}
   bufer = 0
   producerCond = {}
   consumerCond = {}

2. *C1 prosi o 5*
   Lock = {P1, P2, C2}
   bufer = 0
   producerCond = {}
   consumerCond = {C1(5)}

3. *C2 prosi o 3*
   Lock = {P1, P2}
   bufer = 0
   producerCond = {}
   consumerCond = {C1(5), C2(3)}

4. *P1 produkuje 2 i budzi C1*
   Lock = {P1, P2, C1(5)}
   bufer = 2
   producerCond = {}
   consumerCond = {C2(3)}

5. *C1 nadal nie może skonsumować*
   Lock = {P1, P2}
   bufer = 2
   producerCond = {}
   consumerCond = {C1(5),C2(3)}

6. *P1 produkuje 1 i budzi C2*
   Lock = {P1, P2, C2(3)}
   bufer = 3
   producerCond = {}
   consumerCond = {C1(5)}

7. *C2 konsumuje 3*
   Lock = {P1, P2, C2}
   bufer = 0
   producerCond = {}
   consumerCond = {C1(5)}

Dalej możemy powtarzać kroki [3, 7] i doprowadzić do zagłodzenia wątku C1.

Przykład W mojej implementacji maksymalny rozmiar bufora to 50. Wszystkie wątki producenta jednorazową produkują [1, 25] elementów. Wszystkie wątki konsumenta (oprócz pierwszego) domagają się po 1 elemencie. Natomiast pierwszy wątek konsumenta zawsze prosi o 25. Dodatkowo wywoływana jest funkcja Thread.sleep() po każdej produkcji/konsumpcji.

```
Round 1
Consumer 1 consumed 0 times
Consumer 2 consumed 0 times
Consumer 3 consumed 0 times
Consumer 4 consumed 0 times

Round 2
Consumer 1 consumed 17 times
Consumer 2 consumed 200 times
Consumer 3 consumed 279 times
Consumer 4 consumed 637 times

Round 3
Consumer 1 consumed 23 times
Consumer 2 consumed 609 times
Consumer 3 consumed 780 times
Consumer 4 consumed 1097 times

Round 4
Consumer 1 consumed 25 times
Consumer 2 consumed 1092 times
Consumer 3 consumed 1327 times
Consumer 4 consumed 1727 times

Round 5
Consumer 1 consumed 32 times
Consumer 2 consumed 1624 times
Consumer 3 consumed 1883 times
Consumer 4 consumed 2246 times
```

Zrzut ekranu 1: Zagłodzenie wątku 1

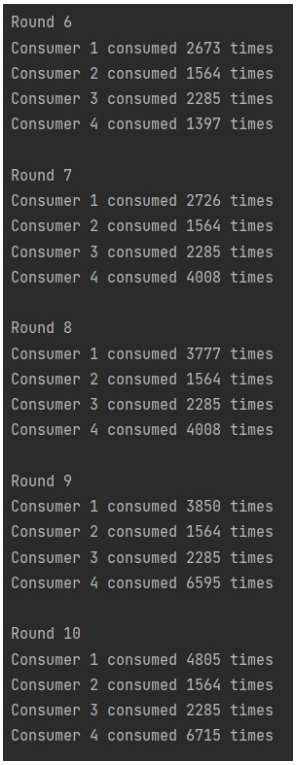
```
Round 6
Consumer 1 consumed 38 times
Consumer 2 consumed 2173 times
Consumer 3 consumed 2407 times
Consumer 4 consumed 2486 times

Round 7
Consumer 1 consumed 59 times
Consumer 2 consumed 2604 times
Consumer 3 consumed 3394 times
Consumer 4 consumed 2710 times

Round 8
Consumer 1 consumed 82 times
Consumer 2 consumed 3585 times
Consumer 3 consumed 3452 times
Consumer 4 consumed 3398 times

Round 9
Consumer 1 consumed 92 times
Consumer 2 consumed 4386 times
Consumer 3 consumed 3520 times
Consumer 4 consumed 3981 times

Round 10
Consumer 1 consumed 106 times
Consumer 2 consumed 4974 times
Consumer 3 consumed 3785 times
Consumer 4 consumed 4409 times
```

Zrzut ekranu 2: Zagłodzenie wątku 1

Dla tych samych warunków, ale bufora zaimplementowanego na czterech zmiennych Condition, sytuacja wygląda następujaco:

```
Round 1
Consumer 1 consumed 0 times
Consumer 2 consumed 0 times
Consumer 3 consumed 1 times
Consumer 4 consumed 0 times

Round 2
Consumer 1 consumed 424 times
Consumer 2 consumed 439 times
Consumer 3 consumed 239 times
Consumer 4 consumed 73 times

Round 3
Consumer 1 consumed 1098 times
Consumer 2 consumed 439 times
Consumer 3 consumed 316 times
Consumer 4 consumed 778 times

Round 4
Consumer 1 consumed 1856 times
Consumer 2 consumed 439 times
Consumer 3 consumed 323 times
Consumer 4 consumed 1396 times

Round 5
Consumer 1 consumed 2223 times
Consumer 2 consumed 439 times
Consumer 3 consumed 2048 times
Consumer 4 consumed 1396 times
```

Zrzut ekranu 3: Brak zagłodzenia



Zrzut ekranu 4: Brak zagłodzenia

2 Zagłodzenie-hasWaiters()

Założenia takie same jak w wcześniejszym punkcie
Układ zdażeń:

1. *Start*
   Lock = {P1, P2, C1, C2}
   bufer = 0
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {}
   consumerCond = {}

2. *C1 prosi o 5*
   Lock = {P1, P2, C2}
   bufer = 0
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C1(5)}
   consumerCond = {}

3. *P1 produkuje 1 i sygnalizuję to C1 (ale C1 nie dostaję Locka)*
   Lock = {P1, P2, C2, C1(5)}
   bufer = 1
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {}
   consumerCond = {}

4. *C2 prosi o 3*
   Lock = {P1, P2, C1(5)}
   bufer = 1
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C2(3)}
   consumerCond = {}

5. *C1 dostaję Locka*
   Lock = {P1, P2)}
   bufer = 1
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C2(3), C1(5)}
   consumerCond = {}

6. *P1 produkuje 2 i sygnalizuje to C2*
   Lock = {P1, P2, C2(3)}
   bufer = 3
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C1(5)}
   consumerCond = {}

7. *C2 konsumuje 3*
   Lock = {P1, P2, C2}
   bufer = 0
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C1(5)}
   consumerCond = {}

Następnie możemy wykonywać kroki [4, 7]. W takiej sytacji zagłodzony zostanie wątek C1. Dzieje się tak, gdyż przy użyciu funkcji hasWaiters() istnieje przypadek, w którym wpuszczamy do kolejki first więcej niż jeden wątek. Traci wówczas ona swoje zalety (brak zagłodzenia)

3 Zakleszczenie - hasWaiters()

Założenia takie same jak w poprzednim punkcie.
Układ zdażen:

1. *Start*
   Lock = {P1, P2, C1, C2}
   bufer = 0
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {}
   consumerCond = {}

2. *C1 prosi o 5*
   Lock = {P1, P2, C2}
   bufer = 0
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C1(5)}
   consumerCond = {}

3. *P1 produkuje 1 i sygnalizuje do C1(nie dostaje Locka)*
   Lock = {P1, P2, C2, C1(5)}

bufer = 1
firstproducerCond = {}
producerCond = {}
firstconsumerCond = {}
consumerCond = {}

4. *C2 prosi o 2*
   Lock = {P1, P2, C1(5)}
   bufer = 1
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C2(2)}
   consumerCond = {}

5. *C1 dostaję Locka, ale nadal nie może skonsumować*
   Lock = {P1, P2)}
   bufer = 1
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C1(5), C2(2)}
   consumerCond = {}

6. *P1 produkuje 3 i sygnalizuje to C1*
   Lock = {P1, P2, C1(5))}
   bufer = 4
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C2(2)}
   consumerCond = {}

7. *C1 dostaję Locka, ale nadal nie może skonsumować*
   Lock = {P1, P2)}
   bufer = 4
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C1(5), C2(2)}
   consumerCond = {}

8. *P1 produkuje 5 i sygnalizuje to C2(nie dostaję Locka)*
   Lock = {P1, P2, C2(2)}
   bufer = 9
   firstproducerCond = {}
   producerCond = {}
   firstconsumerCond = {C1(5))}
   consumerCond = {}

9. *P1 próbuje wyprodukować 5*
   Lock = {P2, C2(2)}
   bufer = 9
   firstproducerCond = {P1(5)}
   producerCond = {}
   firstconsumerCond = {C1(5)}
   consumerCond = {}

10. *C2 dostaję Locka, konsumuje 2 i sygnalizuje to P1(nie dostaje Locka)*
    Lock = {P2, C2, P1(5)}
    bufer = 7
    firstproducerCond = {}
    producerCond = {}

firstconsumerCond = {C1(5)}
consumerCond = {}

11. *P2 próbuje wyprodukować 5*
    Lock = {C2, P1(5)}
    bufer = 7
    firstproducerCond = {P2(5)}
    producerCond = {}
    firstconsumerCond = {C1(5)}
    consumerCond = {}

12. *P1 dostaję Locka, ale nadal nie może wyprodukować*
    Lock = {C2}
    bufer = 7
    firstproducerCond = {P1(5), P2(5)}
    producerCond = {}
    firstconsumerCond = {C1(5)}
    consumerCond = {}

13. *C2 próbuje skonsumować 1, ale musi iść do kolejki dla innych*
    Lock = {}
    bufer = 7
    firstproducerCond = {P1(5), P2(5)}
    producerCond = {}
    firstconsumerCond = {C1(5)}
    consumerCond = {C2(1)}

W tej sytuacji dochodzi do zakleszczenia. Każdy wątek znajduję się w jakiejś kolejce, i nie ma wątków, które mogłyby go wybudzić. Gdyby w kolejce Lock'a pojawiły się jakieś nowe wątki C_i lub P_i to i tak przy nawet najmniejszych wymaganiach kończyłyby one w kolejkach dla pozostałych konsumentów/producentów. W takiej sytaucji nie miałyby również okazji, aby kogoś wybudzić.