SOI zad. 3 - testowanie poprawności

Testowanie poprawności polega na uruchomieniu programu w tylu konfiguracjach wątków (ilości wątków danego typu), że wyczerpane zostaną wszystkie możliwe scenariusze dostępu do bufora.

Testy z kolejką początkowo pustą

1 wątek `A1`

Gdy w kolejce znajdzie się 10 elementów, to warunek produkcji wątku `A1`, przestanie być spełniony, więc nie wyjdzie on z sekcji krytycznej, lecz odda ją innemu wątkowi - który nie istnieje, czyli nigdy tej sekcji krytycznej nie odzyska i wystąpi zakleszczenie.

```
### PUT (A1 0) -> 1[0,] ###

### PUT (A1 2) -> 2[0,2,] ###

### PUT (A1 4) -> 3[0,2,4,] ###

### PUT (A1 6) -> 4[0,2,4,6,] ###

### PUT (A1 8) -> 5[0,2,4,6,8,] ###

### PUT (A1 10) -> 6[0,2,4,6,8,10,] ###

### PUT (A1 12) -> 7[0,2,4,6,8,10,12,] ###

### PUT (A1 14) -> 8[0,2,4,6,8,10,12,14,] ###

### PUT (A1 16) -> 9[0,2,4,6,8,10,12,14,16,] ###

### PUT (A1 18) -> 10[0,2,4,6,8,10,12,14,16,18,] ###
```

1 wątek `A2`

Na samym początku warunek produkcji `A2` nie będzie spełniony i wystąpi zakleszczenie

```
olek@Olek-PC:~/studia/soi/ex3$ g++ test.cpp -lpthread -o test2
olek@Olek-PC:~/studia/soi/ex3$ ./test2
```

dowolna ilość `B1` i `B2`, bez żadnego `A1` lub `A2`

W każdym przypadku efekt jest ten sam, wątki konsumujące `B1` i `B2` nie mogą zjadać liczb gdy bufor jest pusty, czyli wystąpi zakleszczenie.

```
olek@Olek-PC:~/studia/soi/ex3$ g++ test.cpp -lpthread -o test2
olek@Olek-PC:~/studia/soi/ex3$ ./test2
```

1 wątek `A1`, 1 `A2`

Wystąpi zakleszczenie gdy warunki `A1` i `A2` przestaną być spełnione, czyli gdy w kolejce będzie 20 elementów.

```
### PUT (A1 0) -> 1[0,] ###
### PUT (A2 1) -> 2[0,1,] ###
### PUT (A1 2) -> 3[0,1,2,] ###
### PUT (A2 3) -> 4[0,1,2,3,] ###
### PUT (A1 4) -> 5[0,1,2,3,4,] ###
### PUT (A2 5) -> 6[0,1,2,3,4,5,] ###
### PUT (A1 6) -> 7[0,1,2,3,4,5,6,] ###
### PUT (A2 7) -> 8[0,1,2,3,4,5,6,7,] ###
### PUT (A1 8) -> 9[0,1,2,3,4,5,6,7,8,] ###
### PUT (A2 9) -> 10[0,1,2,3,4,5,6,7,8,9,] ###
### PUT (A1 10) -> 11[0,1,2,3,4,5,6,7,8,9,10,] ###
### PUT (A2 11) -> 12[0,1,2,3,4,5,6,7,8,9,10,11,] ###
### PUT (A1 12) -> 13[0,1,2,3,4,5,6,7,8,9,10,11,12,] ###
### PUT (A2 13) -> 14[0,1,2,3,4,5,6,7,8,9,10,11,12,13,] ###
### PUT (A1 14) -> 15[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,] ###
### PUT (A2 15) -> 16[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,] ###
### PUT (A1 16) -> 17[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,] ###
### PUT (A2 17) -> 18[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,] ###
### PUT (A1 18) -> 19[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,] ###
### PUT (A2 19) -> 20[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,] ###
```

1 wątek `A1`, 1 `B1`

Wątek `A1` produkuje parzyste, a `B1` konsumuje parzyste, więc zakleszczenie nie wystąpi i program będzie się wykonywał poprawnie.

```
### GET (B1) -> 9[40,42,44,46,48,0,2,4,6,] ###
### PUT (A1 8) -> 10[40,42,44,46,48,0,2,4,6,8,] ###
### GET (B1) -> 9[42,44,46,48,0,2,4,6,8,] ###
### PUT (A1 10) -> 10[42,44,46,48,0,2,4,6,8,10,] ###
### GET (B1) -> 9[44,46,48,0,2,4,6,8,10,] ###
### PUT (A1 12) -> 10[44,46,48,0,2,4,6,8,10,12,] ###
### GET (B1) -> 9[46,48,0,2,4,6,8,10,12,] ###
### PUT (A1 14) -> 10[46,48,0,2,4,6,8,10,12,14,] ###
### GET (B1) -> 9[48,0,2,4,6,8,10,12,14,] ###
### PUT (A1 16) -> 10[48,0,2,4,6,8,10,12,14,16,] ###
### GET (B1) -> 9[0,2,4,6,8,10,12,14,16,] ###
### PUT (A1 18) -> 10[0,2,4,6,8,10,12,14,16,18,] ###
### GET (B1) -> 9[2,4,6,8,10,12,14,16,18,] ###
### PUT (A1 20) -> 10[2,4,6,8,10,12,14,16,18,20,] ###
### GET (B1) -> 9[4,6,8,10,12,14,16,18,20,] ###
### PUT (A1 22) -> 10[4,6,8,10,12,14,16,18,20,22,] ###
### GET (B1) -> 9[6,8,10,12,14,16,18,20,22,] ###
### PUT (A1 24) -> 10[6,8,10,12,14,16,18,20,22,24,] ###
### GET (B1) -> 9[8,10,12,14,16,18,20,22,24,] ###
### PUT (A1 26) -> 10[8,10,12,14,16,18,20,22,24,26,] ###
### GET (B1) -> 9[10,12,14,16,18,20,22,24,26,] ###
### PUT (A1 28) -> 10[10,12,14,16,18,20,22,24,26,28,] ###
### GET (B1) -> 9[12,14,16,18,20,22,24,26,28,] ###
### PUT (A1 30) -> 10[12,14,16,18,20,22,24,26,28,30,] ###
```

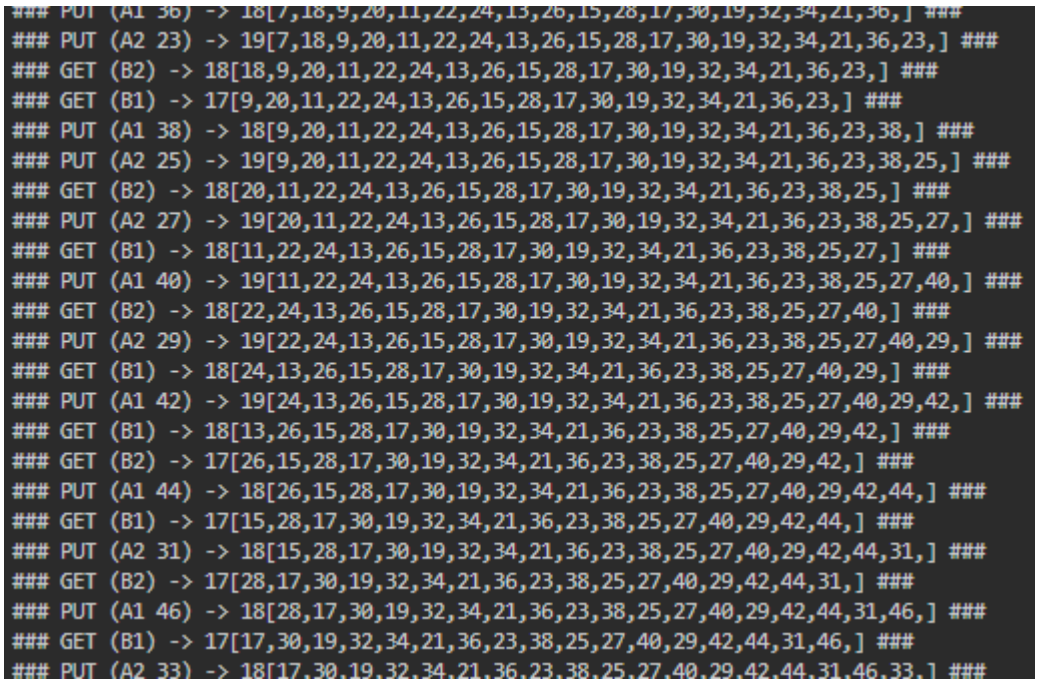
1 wątek `A2`, 1 `B2`

Mimo, że `A2` produkuje nieparzyste, a `B2` je konsumuje, to na samym początku wystąpi zakleszczenie z tego powodu, że warunek `A2` na starcie nie jest spełniony.

```
olek@Olek-PC:~/studia/soi/ex3$ g++ test.cpp -lpthread -o test2
olek@Olek-PC:~/studia/soi/ex3$ ./test2
```

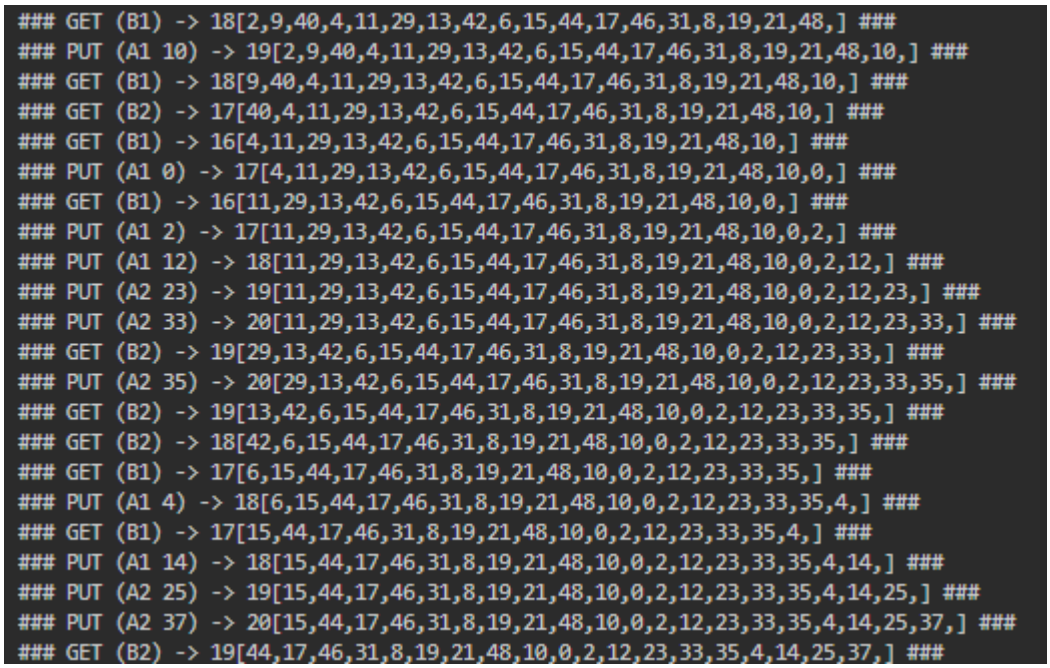
1 wątek `A1`, 1 `A2`, 1 `B1`, 1 `B2`

Program wykonuje się poprawnie bez zakleszczeń i nieprawidłowych dostępów do bufora.



2 wątki `A1`, 2 `A2`, 2 `B1`, 2 `B2`

Program wykonuje się poprawnie bez zakleszczeń i nieprawidłowych dostępów do bufora. Zwiększenie ilości wątków dla obojętnie której klasy nie zakłóca poprawnego działania programu.



Poniżej przykład z: 1 `A1`, 1 `A2`, 10 `B1`, 6 `B2`

```
### GET (B2) -> 7[2,45,4,47,6,8,49,] ###
### GET (B1) -> 6[45,4,47,6,8,49,] ###
### PUT (A1 10) -> 7[45,4,47,6,8,49,10,] ###
### GET (B2) -> 6[4,47,6,8,49,10,] ###
### GET (B1) -> 5[47,6,8,49,10,] ###
### PUT (A2 1) -> 6[47,6,8,49,10,1,] ###
### PUT (A1 12) -> 7[47,6,8,49,10,1,12,] ###
### GET (B2) -> 6[6,8,49,10,1,12,] ###
### GET (B1) -> 5[8,49,10,1,12,] ###
### PUT (A2 3) -> 6[8,49,10,1,12,3,] ###
### GET (B1) -> 5[49,10,1,12,3,] ###
### PUT (A1 14) -> 6[49,10,1,12,3,14,] ###
### PUT (A1 16) -> 7[49,10,1,12,3,14,16,] ###
### PUT (A2 5) -> 8[49,10,1,12,3,14,16,5,] ###
### GET (B2) -> 7[10,1,12,3,14,16,5,] ###
### GET (B1) -> 6[1,12,3,14,16,5,] ###
### PUT (A1 18) -> 7[1,12,3,14,16,5,18,] ###
```

Testy z kolejką początkowo pełną (10 el.)

W zadaniu kolejka nie ma górnego ograniczenia posiadanych elementów, więc na potrzeby tych testów jest 'pełna' wtedy, gdy `A1` nie może generować nowych liczb, czyli ma 10 liczb parzystych. Kolejka na początek ma więc następujące wartości `[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]`

1 wątek `A1`

Warunek `A1` na początku nie jest spełniony, czyli nastąpi zakleszczenie.

```
olek@Olek-PC:~/studia/soi/ex3$ g++ ./test.cpp -o test -lpthread
olek@Olek-PC:~/studia/soi/ex3$ ./test
^C
```

1 wątek `A2`

Tym razem, na samym początku warunek `A2` jest spełniony, więc wątek wygeneruje 10 liczb, po czym warunek generowania nie będzie już dłużej spełniony, więc zakleszczy się.

```
olek@Olek-PC:~/studia/soi/ex3$ ./test
### PUT (A2 1) -> 11[0,2,4,6,8,10,12,14,16,18,1,] ###
### PUT (A2 3) -> 12[0,2,4,6,8,10,12,14,16,18,1,3,] ###
### PUT (A2 5) -> 13[0,2,4,6,8,10,12,14,16,18,1,3,5,] ###
### PUT (A2 7) -> 14[0,2,4,6,8,10,12,14,16,18,1,3,5,7,] ###
### PUT (A2 9) -> 15[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,] ###
### PUT (A2 11) -> 16[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,] ###
### PUT (A2 13) -> 17[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,] ###
### PUT (A2 15) -> 18[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,15,] ###
### PUT (A2 17) -> 19[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,15,17,] ###
### PUT (A2 19) -> 20[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,15,17,19,] ###

^C
```

1 wątek `B1`

Wątek `B1` zjada liczby parzyste do momentu aż zostanie ich < 3. I wtedy się zakleszczy.

```
olek@Olek-PC:~/studia/soi/ex3$ ./test
### GET (B1) -> 9[2,4,6,8,10,12,14,16,18,] ###
### GET (B1) -> 8[4,6,8,10,12,14,16,18,] ###
### GET (B1) -> 7[6,8,10,12,14,16,18,] ###
### GET (B1) -> 6[8,10,12,14,16,18,] ###
### GET (B1) -> 5[10,12,14,16,18,] ###
### GET (B1) -> 4[12,14,16,18,] ###
### GET (B1) -> 3[14,16,18,] ###
### GET (B1) -> 2[16,18,] ###

^C
```

1 wątek `B2`

Warunek wątku od samego początku nie będzie spełniony, ponieważ bufor zawiera same liczby parzyste - wątek `B2` zakleszczy się.

```
olek@Olek-PC:~/studia/soi/ex3$ g++ ./test.cpp -o test -lpthread
olek@Olek-PC:~/studia/soi/ex3$ ./test
^C
```

1 wątek `A1`, 1 `A2`

Sytuacja taka sama gdy mamy tylko wątek `A2`, ponieważ warunek `A1` będzie od samego początku nie spełniony.

```
olek@Olek-PC:~/studia/soi/ex3$ ./test
### PUT (A2 1) -> 11[0,2,4,6,8,10,12,14,16,18,1,] ###
### PUT (A2 3) -> 12[0,2,4,6,8,10,12,14,16,18,1,3,] ###
### PUT (A2 5) -> 13[0,2,4,6,8,10,12,14,16,18,1,3,5,] ###
### PUT (A2 7) -> 14[0,2,4,6,8,10,12,14,16,18,1,3,5,7,] ###
### PUT (A2 9) -> 15[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,] ###
### PUT (A2 11) -> 16[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,] ###
### PUT (A2 13) -> 17[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,] ###
### PUT (A2 15) -> 18[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,15,] ###
### PUT (A2 17) -> 19[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,15,17,] ###
### PUT (A2 19) -> 20[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,15,17,19,] ###

^C
```

1 wątek `A1`, 1 `B1`

Sytuacja podobna jak w przypadku pustej kolejki przy inicjalizacji, różnica jest taka, że w tym przypadku to wątek `B1` uzyska na samym początku dostęp do kolejki, ponieważ warunek dla `A1` z początku nie jest spełniony.

```
### PUT (A1 14) -> 10[46,48,0,2,4,6,8,10,12,14,] ###
### GET (B1) -> 9[48,0,2,4,6,8,10,12,14,] ###
### PUT (A1 16) -> 10[48,0,2,4,6,8,10,12,14,16,] ###
### GET (B1) -> 9[0,2,4,6,8,10,12,14,16,] ###
### PUT (A1 18) -> 10[0,2,4,6,8,10,12,14,16,18,] ###
### GET (B1) -> 9[2,4,6,8,10,12,14,16,18,] ###
### PUT (A1 20) -> 10[2,4,6,8,10,12,14,16,18,20,] ###
### GET (B1) -> 9[4,6,8,10,12,14,16,18,20,] ###
### PUT (A1 22) -> 10[4,6,8,10,12,14,16,18,20,22,] ###
### GET (B1) -> 9[6,8,10,12,14,16,18,20,22,] ###
### PUT (A1 24) -> 10[6,8,10,12,14,16,18,20,22,24,] ###
### GET (B1) -> 9[8,10,12,14,16,18,20,22,24,] ###
### PUT (A1 26) -> 10[8,10,12,14,16,18,20,22,24,26,] ###
### GET (B1) -> 9[10,12,14,16,18,20,22,24,26,] ###
### PUT (A1 28) -> 10[10,12,14,16,18,20,22,24,26,28,] ###
```

1 wątek `A2`, 1 `B1`

Przez to, że `A2` produkuje l. nieparzyste, a `B1` konsumuje tylko parzyste, to gdy w buforze skończą się liczby parzyste, to wystąpi zakleszczenie.

```
olek@Olek-PC:~/studia/soi/ex3$ ./test
### PUT (A2 1) -> 11[0,2,4,6,8,10,12,14,16,18,1,] ###
### GET (B1) -> 10[2,4,6,8,10,12,14,16,18,1,] ###
### PUT (A2 3) -> 11[2,4,6,8,10,12,14,16,18,1,3,] ###
### GET (B1) -> 10[4,6,8,10,12,14,16,18,1,3,] ###
### PUT (A2 5) -> 11[4,6,8,10,12,14,16,18,1,3,5,] ###
### GET (B1) -> 10[6,8,10,12,14,16,18,1,3,5,] ###
### PUT (A2 7) -> 11[6,8,10,12,14,16,18,1,3,5,7,] ###
### GET (B1) -> 10[8,10,12,14,16,18,1,3,5,7,] ###
### PUT (A2 9) -> 11[8,10,12,14,16,18,1,3,5,7,9,] ###
### GET (B1) -> 10[10,12,14,16,18,1,3,5,7,9,] ###
### GET (B1) -> 9[12,14,16,18,1,3,5,7,9,] ###
### GET (B1) -> 8[14,16,18,1,3,5,7,9,] ###
### GET (B1) -> 7[16,18,1,3,5,7,9,] ###
### GET (B1) -> 6[18,1,3,5,7,9,] ###
### GET (B1) -> 5[1,3,5,7,9,] ###

^C
```

1 wątek `A2`, 1 `B2`

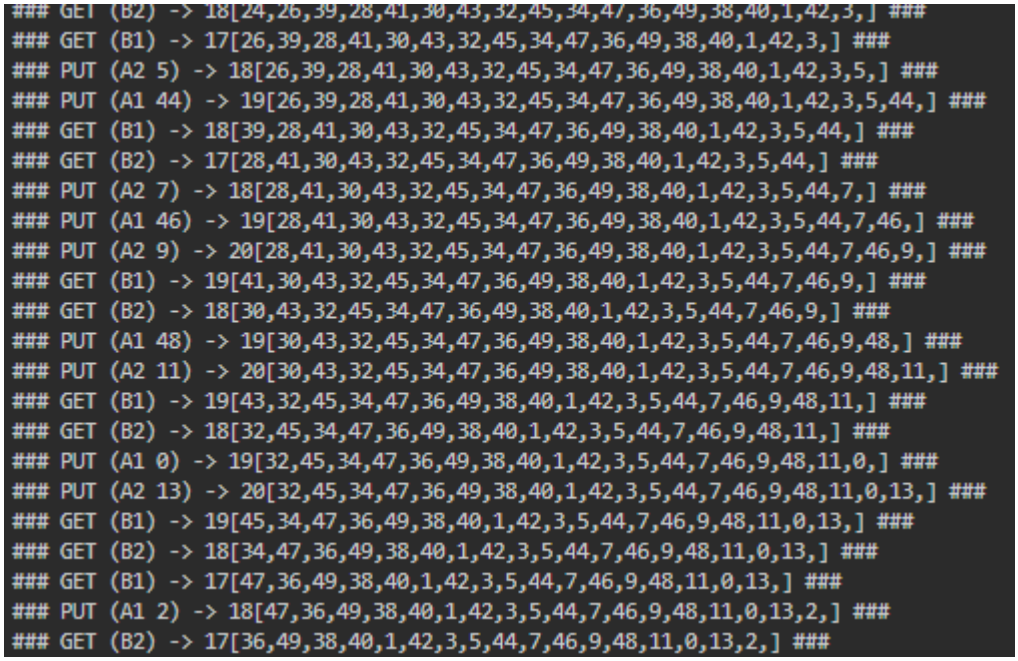
mimo, że `A2` produkuje l. nieparzyste, a `B2` konsumuje nieparzyste, to i tak wystąpi zakleszczenie, spowodowane tym, że `B2` może konsumować tylko z początku kolejki, a tam znajdują się liczby parzyste, którymi została ona zainicjalizowana.

```
olek@Olek-PC:~/studia/soi/ex3$ ./test
### PUT (A2 1) -> 11[0,2,4,6,8,10,12,14,16,18,1,] ###
### PUT (A2 3) -> 12[0,2,4,6,8,10,12,14,16,18,1,3,] ###
### PUT (A2 5) -> 13[0,2,4,6,8,10,12,14,16,18,1,3,5,] ###
### PUT (A2 7) -> 14[0,2,4,6,8,10,12,14,16,18,1,3,5,7,] ###
### PUT (A2 9) -> 15[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,] ###
### PUT (A2 11) -> 16[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,] ###
### PUT (A2 13) -> 17[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,] ###
### PUT (A2 15) -> 18[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,15,] ###
### PUT (A2 17) -> 19[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,15,17,] ###
### PUT (A2 19) -> 20[0,2,4,6,8,10,12,14,16,18,1,3,5,7,9,11,13,15,17,19,] ###

^C
```

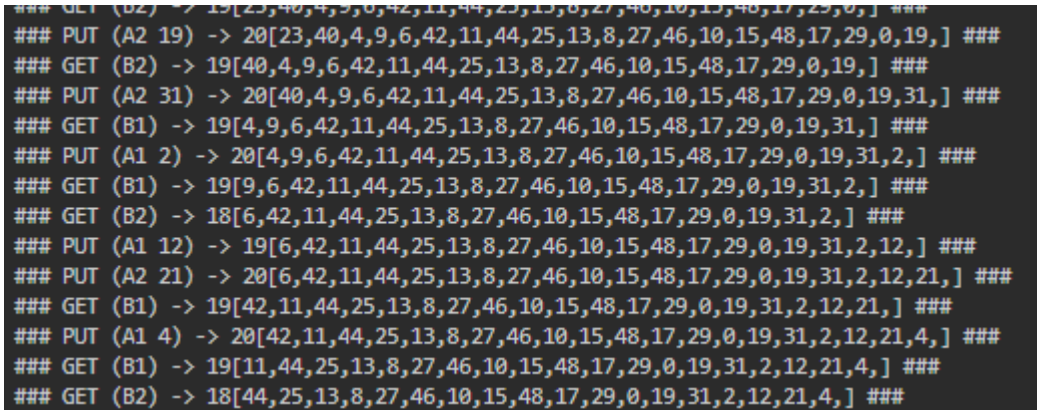
1 wątek `A1`, 1 `A2`, 1 `B1`, 1 `B2`

Program wykonuje się poprawnie bez zakleszczeń i nieprawidłowych dostępów do bufora.



2 wątki `A1`, 2 `A2`, 2 `B1`, 2 `B2`

Program wykonuje się poprawnie bez zakleszczeń i nieprawidłowych dostępów do bufora. Zwiększenie ilości wątków dla obojętnie której klasy nie zakłóca poprawnego działania programu.



Poniżej przykład z: 1 `A1`, 1 `A2`, 10 `B1`, 6 `B2`