Programowanie współbieżne

Wzajemne wykluczanie

algorytmy Dekkera, Petersona, Lamporta

Prowadzący: dr inż. Jarosław Rulka
jaroslaw.rulka@wat.edu.pl

Blokowanie procesu współbieżnego

- Występują dwa sposoby oczekiwania procesu współbieżnego na udostępnienie współdzielonego zasobu:

  - **aktywne czekanie** (ang. busy-waiting) – proces samoczynnie co pewien interwał czasowy ponawia swoje żądanie dostępu do zasobu:
    - **trwałe aktywne czekanie** – proces ponawia żądania dostępu aż do skutecznego wywłaszczenia zasobu,
    - **nietrwałe aktywne czekanie** – proces ponawia żądanie dostępu do zasobu określoną liczbę razy, po czym przechodzi do pasywnego czekania.

  - **pasywne czekanie** – proces jednorazowo wysyła do tzw. „planisty” żądanie dostępu do zasobu, po czy przestaje być aktywny. Jedynie planista jest w stanie ponownie uaktywnić ten proces.

Własność uczciwości – rodzaje

- ***Uczciwość słaba*** – jeżeli proces nieprzerwanie zgłasza żądanie, to w końcu będzie ono obsłużone.
- ***Uczciwość mocna*** – jeżeli proces zgłasza żądanie nieskończenie wiele razy, to w końcu będzie ono obsłużone.
- ***Oczekiwanie liniowe*** – jeżeli proces zgłasza żądanie, to będzie ono obsłużone zanim dowolny inny proces zostanie obsłużony więcej niż raz.
- ***FIFO*** (pierwszy wszedł, pierwszy wyjdzie) – jeżeli proces zgłasza żądanie, to będzie ono obsłużone przed dowolnym żądaniem zgłoszonym później.

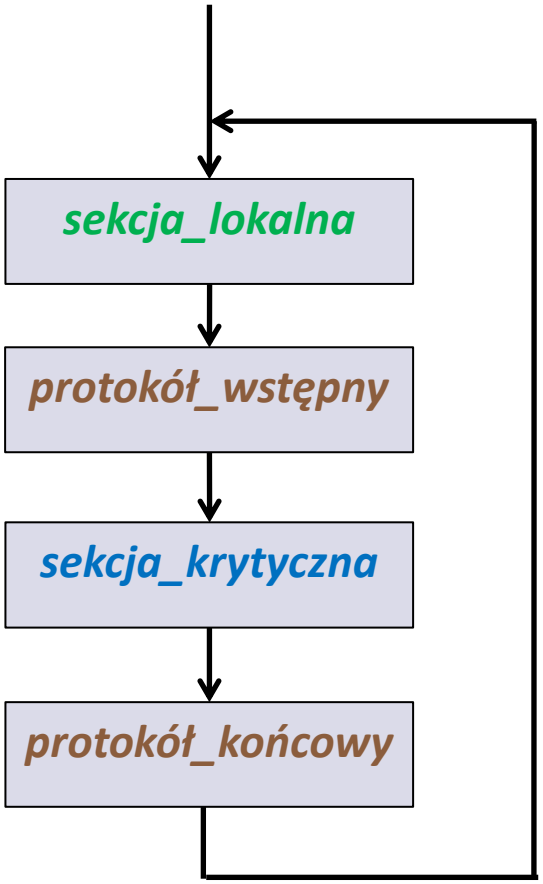
- Uczciwość słaba i mocna mają znaczenie teoretyczne.
- W praktyce jest stosowane oczekiwanie liniowe lub FIFO.
- Obydwa mogą być implementowane w systemach scentralizowanych – w systemach rozproszonych występują problemy z realizacją algorytmu FIFO.

Problem wzajemnego wykluczania

- Problem wzajemnego wykluczania można zdefiniować jako:
  - Zsynchronizować N procesów, z których każdy, w pętli realizuje:
    - sekcję lokalną (tzw. sprawy własne)
    - potem protokół wstępny/sekcję wejściową (często oparty na pętli aktywnego czekania)
    - następnie działania na części współdzielonej – sekcja krytyczna,
    - i kończy iterację protokołem końcowym/sekcją wyjściową;
- Należy zapewnić spełnienie zasad:
  - W sekcji krytycznej może przebywać co najwyżej jeden proces jednocześnie (własność bezpieczeństwa);
  - Każdy proces, który chce wykonać sekcję krytyczną, w skończonym czasie powinien do niej wejść (własność żywotności) – nie wystąpi **zagłodzenie**.

Problem wzajemnego wykluczania

- Schemat pracy procesów współbieżnych wyglądać może następująco:




```
1. begin
2.    loop
3.    begin
4.       sekcja_lokalna;
5.       protokół_wstępny;
6.       sekcja_krytyczna;
7.       protokół_końcowy;
8.    end loop;
9. end;
```

Problem wzajemnego wykluczania – założenia (1/2)

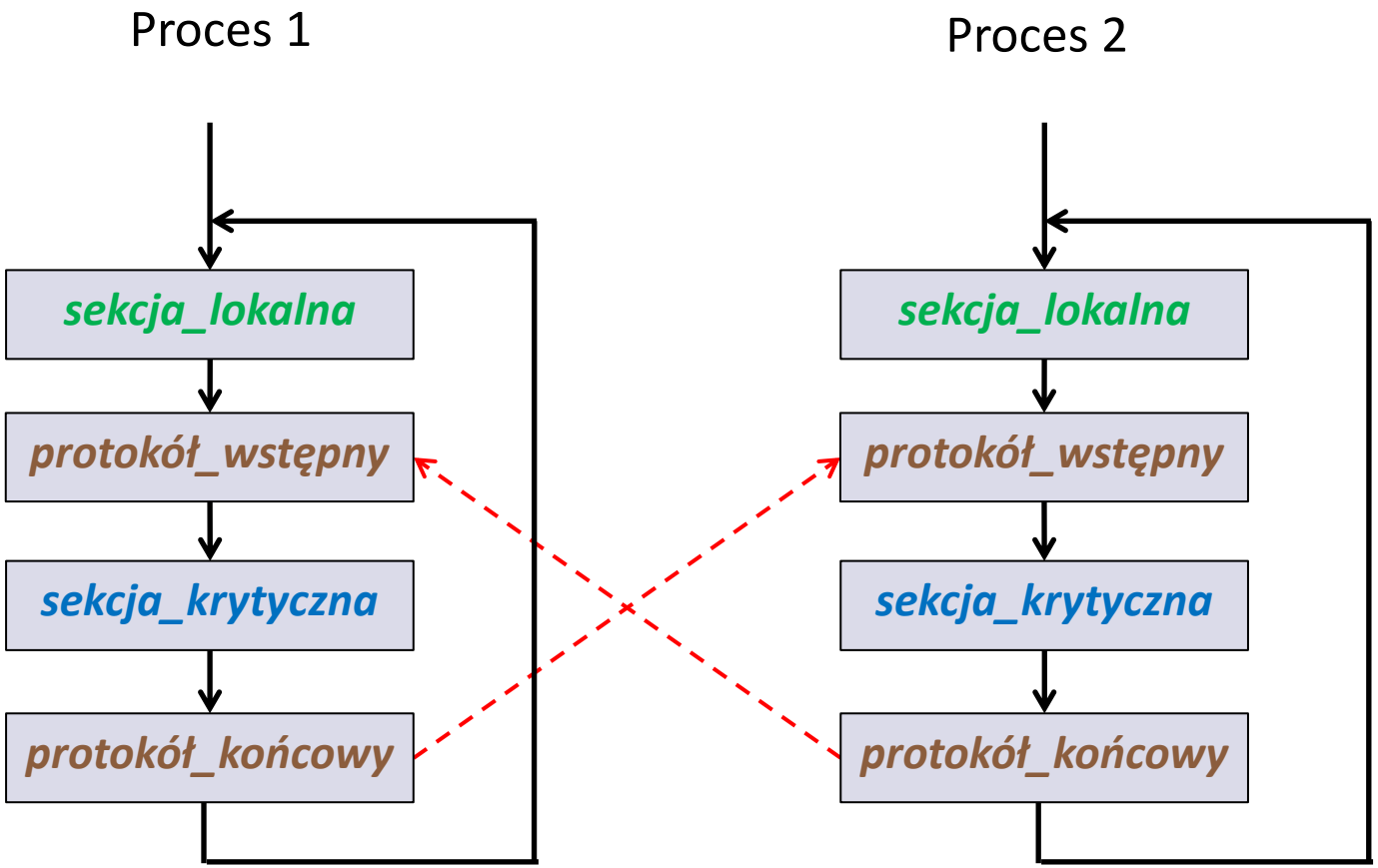
- Dalsze założenia w rozwiązaniu wzajemnego wykluczania:
  a. Procesy muszą być traktowane jako równoważne (brak priorytetów);
  b. Wynik działania programu współbieżnego nie może zależeć od przebiegu przeplotu (przełączania procesów);
  c. Wynik działania musi być poprawny dla wszystkich możliwych dopuszczalnych przeplotów;
  d. Jeśli nie będzie rywalizacji o wejście do sekcji krytycznej, to proces, który zażąda wejścia, zrealizuje to wejście;
  e. Program musi spełniać własność wzajemnego wykluczania, tzn. instrukcje z sekcji krytycznych dwu lub więcej procesów nie mogą się wykonywać jednocześnie;

Problem wzajemnego wykluczania – założenia (2/2)

f. Żaden proces nie może wykonywać swej sekcji krytycznej nieskończenie długo (nie może także ulec awarii podczas wykonywania sekcji krytycznej). Taki sam warunek dotyczy protokołów wstępnego i końcowego;
g. Jeżeli więcej niż jeden proces chce wejść do sekcji krytycznej, to decyzja o tym, który proces zostanie wybrany musi być podjęta w skończonym czasie.

- Rola protokołów jest następująca:
  - **wstępnego**: przy wejściu do sekcji krytycznej proces sprawdza, czy może wejść do sekcji, w przypadku niespełnienia warunku proces jest blokowany/wstrzymywany;
  - **końcowego**: po wyjściu z sekcji informuje inne procesy, iż opuścił sekcję krytyczną, zatem inny proces może wejść do sekcji;

Algorytm – schemat



Czynności protokołu końcowego wpływają na warunki sprawdzane w protokole wstępnym

Wzajemne wykluczanie – algorytmy naiwne

Wzajemne wykluczanie – próba nr 1

```
1.  czyja_kol : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;
6.        while czyja_kol = 2
7.        end while;
8.        Sekcja_krytyczna_1;
9.        czyja_kol := 2;
10.    end loop;
11. end P1;
```

```
12. process P2 is
13. begin
14.    loop
15.       Sekcja_lokalna_2;
16.       while czyja_kol = 1
17.       end while;
18.       Sekcja_krytyczna_2;
19.       czyja_kol := 1;
20.    end loop;
21. end P2;
```

Uwaga: tu występuje „aktywne oczekiwanie”

Wzajemne wykluczanie – próba nr 1 (symulacja 1/4)

```
1.  czyja_kol : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                6
6.        while czyja_kol = 2              7
7.        end while;
8.        Sekcja_krytyczna_1;              1
9.        czyja_kol := 2;                  2
10.    end loop;
11. end P1;
```

```
12. process P2 is
13. begin
14.    loop
15.       Sekcja_lokalna_2;                3
16.       while czyja_kol = 1              4
17.       end while;
18.       Sekcja_krytyczna_2;              5
19.       czyja_kol := 1;                  8
20.    end loop;
21. end P2;
```

Wzajemne wykluczanie – próba nr 1 (symulacja 2/4)

```
1.  czyja_kol : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                    6
6.        while czyja_kol = 2                  11|7
7.        end while;
8.        Sekcja_krytyczna_1;                  12|1
9.        czyja_kol := 2;                      2
10.    end loop;
11. end P1;
```

```
12. process P2 is
13. begin
14.    loop
15.       Sekcja_lokalna_2;                    9|3
16.       while czyja_kol = 1                  10|4
17.       end while;
18.       Sekcja_krytyczna_2;                  5
19.       czyja_kol := 1;                      8
20.    end loop;
21. end P2;
```

Wzajemne wykluczanie – próba nr 1 (symulacja 3/4)

```
1.  czyja_kol : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                15|6
6.        while czyja_kol = 2              16|11|7
7.        end while;
8.        Sekcja_krytyczna_1;              12|1
9.        czyja_kol := 2;                  14|2
10.    end loop;
11. end P1;
```

```
12. process P2 is
13. begin
14.    loop
15.       Sekcja_lokalna_2;                9|3
16.       while czyja_kol = 1              13|10|4
17.       end while;
18.       Sekcja_krytyczna_2;              5
19.       czyja_kol := 1;                  8
20.    end loop;
21. end P2;
```

Wzajemne wykluczanie – próba nr 1 (symulacja 4/4)

```
1.  czyja_kol : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                    15|6
6.        while czyja_kol = 2                  20|16|11|7
7.        end while;
8.        Sekcja_krytyczna_1;                  12|1
9.        czyja_kol := 2;                      14|2
10.    end loop;
11. end P1;
```

```
12. process P2 is
13. begin
14.    loop
15.       Sekcja_lokalna_2;                    9|3
16.       while czyja_kol = 1                  17|13|10|4
17.       end while;
18.       Sekcja_krytyczna_2;                  18|5
19.       czyja_kol := 1;                      19|8
20.    end loop;
21. end P2;
```

Wzajemne wykluczanie – próba nr 1

```
1.  czyja_kol : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;
6.        while czyja_kol = 2
7.        end while;
8.        Sekcja_krytyczna_1;
9.        czyja_kol := 2;
10.    end loop;
11. end P1;
```

```
12. process P2 is
13. begin
14.    loop
15.       Sekcja_lokalna_2;
16.       while czyja_kol = 1
17.       end while;
18.       Sekcja_krytyczna_2;
19.       czyja_kol := 1;
20.    end loop;
21. end P2;
```

Pierwsza próba – opis (1/2)

1. Protokół wstępny: linie (P1) 6-7, (P2) 16-17
2. Protokół końcowy: linie (P1) 9, (P2) 19
3. To rozwiązanie zapewnia wzajemne wykluczanie – nigdy dwa procesy nie znajdą się jednocześnie w sekcji krytycznej.
   Zapewnia to zmienna `czyja_kolej` określająca uprawnionego do wejścia do sekcji krytycznej.
4. To rozwiązanie nigdy nie doprowadza do zakleszczenia
   Aby w programie wystąpiło zakleszczenie oba procesy muszą nieskończenie długo testować wartość zmiennej `czyja_kolej`, przy czym warunek musi być stale fałszywy.
5. W tym rozwiązaniu nie może wystąpić zagłodzenie – jeden proces musiałby nieskończenie wiele razy wchodzić do sekcji krytycznej, podczas gdy drugi wciąż wykonywałby swój protokół wstępny.
   Każdy proces w protokole końcowym przekazuje prawo do wejścia do sekcji krytycznej drugiemu procesowi i prawo to uzyska dopiero, gdy drugi proces mu przekaże w swoim protokole końcowym.

Pierwsza próba – opis (2/2)

7. Niestety w przypadku braku współzawodnictwa to rozwiązanie zawodzi.
   - Gdy jeden z procesów utknie w swojej sekcji lokalnej, to drugi proces może wejść do sekcji krytycznej co najwyżej raz.
8. Obydwa procesy są zmuszane do pracy w zbliżonym tempie.
   - Gdy jeden z procesów musi wchodzić bardzo często do sekcji krytycznej (np. co 1 sekundę) a drugi bardzo rzadko (np. raz na godzinę), to takie rozwiązanie nie może być przyjęte. Jeden z procesów spowalnia (blokuje) pracę drugiego.
9. Te wady wykluczają przyjęcie tego rozwiązania.
10. Technika programowania polegająca na jawnym przekazywaniu prawa do wykonywania się nazywana jest **współprogramowaniem**.

Wzajemne wykluczanie – próba nr 2

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.    loop
5.      Sekcja_lokalna_1;
6.      while K2 = 0
7.      end while;
8.      K1 := 0;
9.      Sekcja_krytyczna_1;
10.     K1 := 1;
11.   end loop;
12. end P1;
```

```
13. process P2 is
14. begin
15.   loop
16.     Sekcja_lokalna_2;
17.     while K1 = 0
18.     end while;
19.     K2 := 0;
20.     Sekcja_krytyczna_2;
21.     K2 := 1;
22.   end loop;
23. end P2;
```

Wzajemne wykluczanie – próba nr 2 (symulacja 1/2)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                    1
6.        while K2 = 0                         3
7.        end while;
8.        K1 := 0;                             4
9.        Sekcja_krytyczna_1;                  5
10.       K1 := 1;                             7
11.    end loop;
12. end P1;
```

```
13. process P2 is
14. begin
15.    loop
16.       Sekcja_lokalna_2;                    2
17.       while K1 = 0                         6
18.       end while;
19.       K2 := 0;
20.       Sekcja_krytyczna_2;
21.       K2 := 1;
22.    end loop;
23. end P2;
```

Wzajemne wykluczanie – próba nr 2 (symulacja 2/2)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                    1
6.        while K2 = 0                         3
7.        end while;
8.        K1 := 0;                             4
9.        Sekcja_krytyczna_1;                  5
10.       K1 := 1;                             7
11.    end loop;
12. end P1;
```

```
13. process P2 is
14. begin
15.    loop
16.       Sekcja_lokalna_2;                    2
17.       while K1 = 0                         8|6
18.       end while;
19.       K2 := 0;                             9
20.       Sekcja_krytyczna_2;                  10
21.       K2 := 1;
22.    end loop;
23. end P2;
```

Wzajemne wykluczanie – próba nr 2 (symulacja z błędem)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                    1
6.        while K2 = 0                         3
7.        end while;
8.        K1 := 0;
9.        Sekcja_krytyczna_1;
10.       K1 := 1;
11.    end loop;
12. end P1;
```

```
13. process P2 is
14. begin
15.    loop
16.       Sekcja_lokalna_2;                    2
17.       while K1 = 0                         4
          end while;
19.       K2 := 0;                             6
          Sekcja_krytyczna_2;                  8
          K2 := 1;
       end loop;
23. end P2;
```

Druga próba – opis

1. Proces `Pi` sygnalizuje potrzebę wejścia do sekcji krytycznej nadając zmiennej `Ki` wartość `0`.
2. To rozwiązanie nie spełnia własności wzajemnego wykluczania.
   Uwaga! Własność wzajemnego wykluczania jest podstawową i najłatwiejszą do wykazania cechą programowania współbieżnego i od niej należy rozpoczynać analizę każdego rozwiązania.
3. W tym rozwiązaniu dwa procesy mogą być jednocześnie w sekcji krytycznej. Przykładowy ciąg instrukcji udowadnia taką możliwość:
   a) P1 sprawdza wartość K2 i stwierdza, że K2 = 1.
   b) P2 sprawdza wartość K1 i stwierdza, że K1 = 1.
   c) P1 ustawia wartość K1 na 0.
   d) P2 ustawia wartość K2 na 0.
   e) P1 wchodzi do sekcji krytycznej.
   f) P2 wchodzi do sekcji krytycznej.
4. W celu pokazania, że własność wzajemnego wykluczania nie zachodzi wystarczy podać jeden przeplot instrukcji (*tzw. killer'ski przeplot*).
5. Protokół wstępny: linie (P1) 6-8, (P2) 17-19
6. Protokół końcowy: linie (P1) 10, (P2) 21

Wzajemne wykluczanie – próba nr 3

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.    loop
5.      Sekcja_lokalna_1;
6.      K1 := 0;
7.      while K2 = 0
8.      end while;
9.      Sekcja_krytyczna_1;
10.     K1 := 1;
11.   end loop;
12. end P1;
```

```
13. process P2 is
14. begin
15.   loop
16.     Sekcja_lokalna_2;
17.     K2 := 0;
18.     while K1 = 0
19.     end while;
20.     Sekcja_krytyczna_2;
21.     K2 := 1;
22.   end loop;
23. end P2;
```

Wzajemne wykluczanie – próba nr 3 (symulacja 1/2)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                    1
6.        K1 := 0;                             3
7.        while K2 = 0                         4
8.        end while;
9.        Sekcja_krytyczna_1;                  7
10.       K1 := 1;                             8
11.    end loop;
12. end P1;
```

```
13. process P2 is
14. begin
15.    loop
16.       Sekcja_lokalna_2;                    2
17.       K2 := 0;                             5
18.       while K1 = 0                         6
19.       end while;
20.       Sekcja_krytyczna_2;
21.       K2 := 1;
22.    end loop;
23. end P2;
```

Wzajemne wykluczanie – próba nr 3 (symulacja 2/2)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                    9|1
6.        K1 := 0;                             11|3
7.        while K2 = 0                         15|12|4
8.        end while;
9.        Sekcja_krytyczna_1;                  16|7
10.       K1 := 1;                             17|8
11.    end loop;
12. end P1;
```

```
13. process P2 is
14. begin
15.    loop
16.       Sekcja_lokalna_2;                    2
17.       K2 := 0;                             5
18.       while K1 = 0                         10|6
19.       end while;
20.       Sekcja_krytyczna_2;                  13
21.       K2 := 1;                             14
22.    end loop;
23. end P2;
```

Wzajemne wykluczanie – próba nr 3 (symulacja z blokadą)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;
6.        K1 := 0;
7.        while K2 = 0
8.        end while;
9.        Sekcja_krytyczna_1;
10.       K1 := 1;
11.    end loop;
12. end P1;
```

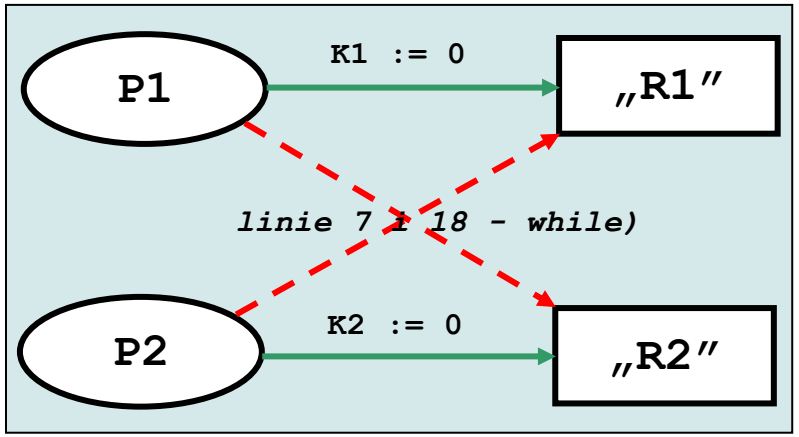
```
13. process P2 is
14. begin
15.    loop
16.       Sekcja_lokalna_2;
17.       K2 := 0;
18.       while K1 = 0
19.       end while;
20.       Sekcja_krytyczna_2;
21.       K2 := 1;
22.    end loop;
23. end P2;
```

Trzecia próba – opis

1. Proces `Pi` sygnalizuje potrzebę wejścia do sekcji krytycznej nadając zmiennej `Ki` wartość `0`.
   - Przypisanie odpowiedniej zmiennej wartości zero sygnalizuje nie tylko chęć wejścia do sekcji krytycznej, ale oznacza także naleganie na przyznanie tego prawa, co niestety może doprowadzić do blokady.
2. To rozwiązanie zapewnia własność wzajemnego wykluczania - czyli nie zdarzy się sytuacja, że dwa procesy będą jednocześnie w sekcji krytycznej.
3. Możliwość wystąpienia zakleszczenia.

   Przykładowy ciąg instrukcji udowadnia taką możliwość:

   a) P1 nadaje zmiennej K1 wartość 0.
   b) P2 nadaje zmiennej K2 wartość 0.
   c) P1 sprawdza wartość K2 i pozostaje w pętli.
   d) P2 sprawdza wartość K1 i pozostaje w pętli.
4. Protokół wstępny: linie (P1) 6-8, (P2) 17-19
5. Protokół końcowy: linie (P1) 10, (P2) 21



Wzajemne wykluczanie – próba nr 4

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.    loop
5.      Sekcja_lokalna_1;
6.      K1 := 0;
7.      while K2 = 0
8.        K1 := 1;
9.        K1 := 0;
10.     end while;
11.     Sekcja_krytyczna_1;
12.     K1 := 1;
13.   end loop;
14. end P1;
```

```
15. process P2 is
16. begin
17.   loop
18.     Sekcja_lokalna_2;
19.     K2 := 0;
20.     while K1 = 0
21.       K2 := 1;
22.       K2 := 0;
23.     end while;
24.     Sekcja_krytyczna_2;
25.     K2 := 1;
26.   end loop;
27. end P2;
```

Wzajemne wykluczanie – próba nr 4 (symulacja zagłodzenia 1/2)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                    1
6.        K1 := 0;                             3
7.        while K2 = 0                         7
8.           K1 := 1;
9.           K1 := 0;
10.       end while;
11.       Sekcja_krytyczna_1;                  8
12.       K1 := 1;                             9
13.   end loop;
14. end P1;
```

```
15. process P2 is
16. begin
17.    loop
18.       Sekcja_lokalna_2;                    2
19.       K2 := 0;                             4
20.       while K1 = 0                         5
21.          K2 := 1;                          6
22.          K2 := 0;
23.       end while;
24.       Sekcja_krytyczna_2;
25.       K2 := 1;
26.    end loop;
27. end P2;
```

Wzajemne wykluczanie – próba nr 4 (symulacja zagłodzenia 2/2)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                10|1
6.        K1 := 0;                         11|3
7.        while K2 = 0                     15|4
8.           K1 := 1;
9.           K1 := 0;
10.       end while;
11.       Sekcja_krytyczna_1;              16|5
12.       K1 := 1;                         17|9
13.    end loop;
14. end P1;
```

```
15. process P2 is
16. begin
17.    loop
18.       Sekcja_lokalna_2;                2
19.       K2 := 0;                         6
20.       while K1 = 0                     13|7
21.          K2 := 1;                      14|8
22.          K2 := 0;                      12
23.       end while;
24.       Sekcja_krytyczna_2;
25.       K2 := 1;
26.    end loop;
27. end P2;
```

Wzajemne wykluczanie – próba nr 4 (symulacja półblokady 1/2)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                    1
6.        K1 := 0;                             3
7.        while K2 = 0                         5
8.           K1 := 1;                          7
9.           K1 := 0;                          9
10.       end while;
11.       Sekcja_krytyczna_1;
12.       K1 := 1;
13.    end loop;
14. end P1;
```

```
15. process P2 is
16. begin
17.    loop
18.       Sekcja_lokalna_2;                    2
19.       K2 := 0;                             4
20.       while K1 = 0                         6
21.          K2 := 1;                          8
22.          K2 := 0;                          10
23.       end while;
24.       Sekcja_krytyczna_2;
25.       K2 := 1;
26.    end loop;
27. end P2;
```

Wzajemne wykluczanie – próba nr 4 (symulacja półblokady 2/2)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;                    1
6.        K1 := 0;                             3
7.        while K2 = 0                         11|5
8.           K1 := 1;                          13|7
9.           K1 := 0;                          15|9
10.       end while;
11.       Sekcja_krytyczna_1;
12.       K1 := 1;
13.    end loop;
14. end P1;
```

```
15. process P2 is
16. begin
17.    loop
18.       Sekcja_lokalna_2;                    2
19.       K2 := 0;                             4
20.       while K1 = 0                         12|6
21.          K2 := 1;                          14|8
22.          K2 := 0;                          16|10
23.       end while;
24.       Sekcja_krytyczna_2;
25.       K2 := 1;
26.    end loop;
27. end P2;
```

Wzajemne wykluczanie – próba nr 4 (symulacja półblokady 2/2)

```
1.  K1, K2 : Integer := 1;

2.  process P1 is
3.  begin
4.     loop
5.        Sekcja_lokalna_1;
6.        K1 := 0;
7.        while K2 = 0
8.           K1 := 1;
9.           K1 := 0;
10.       end while;
11.       Sekcja_krytyczna_1;
12.       K1 := 1;
13.    end loop;
14. end P1;
```

```
15. process P2 is
16. begin
17.    loop
18.       Sekcja_lokalna_2;
19.       K2 := 0;
20.       while K1 = 0
21.          K2 := 1;
22.          K2 := 0;
23.       end while;
24.       Sekcja_krytyczna_2;
25.       K2 := 1;
26.    end loop;
27. end P2;
```

Czwarta próba – opis (1/3)

1. Proces `Pi` sygnalizuje potrzebę wejścia do sekcji krytycznej nadając zmiennej `Ki` wartość `0`.
2. Wprowadzono wymaganie, aby proces po wykryciu współzawodnictwa o wejście do sekcji krytycznej rezygnował na chwilę z tego prawa na rzecz innego procesu.
   - Świadczy o tym ciąg instrukcji typu: `K1 := 1; K1 := 0;` tj. linie 8 i 9 oraz przez analogię linie 21 i 22.
3. To rozwiązanie zapewnia własność wzajemnego wykluczania – czyli nie zdarzy się sytuacja, że dwa procesy będą jednocześnie w sekcji krytycznej.

Czwarta próba – opis (2/3)

- Możliwość zagłodzenia procesu. Przykładowy ciąg instrukcji udowadnia taką możliwość:
  a) (6) P1 przypisuje K1 wartość 0.
  b) (19) P2 przypisuje K2 wartość 0.
  c) (20, 21) P2 sprawdza wartość K1 i przypisuje K2 wartość 1.
  d) P1 wykonuje pełny obrót dużej pętli:
     - (7) sprawdza wartość K2,
     - (11) wychodzi z pętli wewnętrznej i wchodzi do sekcji krytycznej,
     - (12) przywraca wartość 1 zmiennej K1,
     - (5) wykonuje sekcję lokalną,
     - (6) przypisuje zmiennej K1 wartość 0.
  e) (19) P2 przypisuje zmiennej K2 wartość 0.
  f) Jeżeli przejdziemy do kroku c) i będziemy powtarzać tę sekwencję w nieskończoność, to mamy zagłodzenie procesu P2.

Czwarta próba – opis (3/3)

5. W tym rozwiązaniu może wystąpić półblokada (*ang. livelock*).

   W przypadku zakleszczenia nie ma żadnego możliwego ciągu wykonań instrukcji, który doprowadziłby do wejścia do sekcji krytycznej.

   W przypadku półblokady możliwe są obliczenia kończące się sukcesem, ale można też podać jeden lub więcej ciągów wykonań instrukcji, w których żaden proces nigdy nie wejdzie do sekcji krytycznej.

   Przykładowy ciąg instrukcji (przeplot) udowadnia taką możliwość. Zakładamy, że instrukcje obydwu procesów występują naprzemiennie.

   a) P1 przypisuje K1 wartość 0.
   b) P2 przypisuje K2 wartość 0.
   c) P1 bada wartość K2 i pozostaje w pętli.
   d) P2 bada wartość K1 i pozostaje w pętli.
   e) P1 przywraca zmiennej K1 wartość 1.
   f) P2 przywraca zmiennej K2 wartość 1.
   g) P1 przypisuje K1 wartość 0.
   h) P2 przypisuje K2 wartość 0.
   i) P1 bada wartość K2 i pozostaje w pętli.
   j) P2 bada wartość K1 i pozostaje w pętli.

6. Protokół wstępny: linie 6-10, 19-23
7. Protokół końcowy: linie 12, 25

Wzajemne wykluczanie – alg. Dekkera

```
1.  K1, K2 : Integer := 1;
2.  czyja_kolej: Integer := 1;

3.  process P1 is
4.  begin
5.    loop
6.      Sekcja_lokalna_1;
7.      K1 := 0;
8.      while K2 = 0
9.        if czyja_kolej = 2
10.         K1 := 1;
11.         while czyja_kolej = 2
12.         end while;
13.         K1 := 0;
14.       end if;
15.     end while;
16.     Sekcja_krytyczna_1;
17.     K1 := 1;
18.     czyja_kolej := 2;
19.   end loop;
20. end P1;
```

```
21. process P2 is
22. begin
23.   loop
24.     Sekcja_lokalna_2;
25.     K2 := 0;
26.     while K1 = 0
27.       if czyja_kolej = 1
28.         K2 := 1;
29.         while czyja_kolej = 1
30.         end while;
31.         K2 := 0;
32.       end if;
33.     end while;
34.     Sekcja_krytyczna_2;
35.     K2 := 1;
36.     czyja_kolej := 1;
37.   end loop;
38. end P2;
```

Alg. Dekkera (poprawny) – opis (1/2)

1. Algorytm Dekkera (poprawny) jest połączeniem pierwszego i czwartego rozwiązania.
2. W pierwszym podejściu jawnie przekazywano prawo wejścia do sekcji krytycznej, co przy braku współzawodnictwa wykluczało to rozwiązanie.
3. W czwartym podejściu każdy proces miał własną zmienną co zapobiegało brakowi współzawodnictwa.
   Jednak, gdy ono wystąpiło to procesy albo głodziły siebie nawzajem, albo półblokowały.
4. Algorytm Dekkera jest podobny do rozwiązania proponowanego w czwartym podejściu, lecz prawo do nalegania jest jawnie przekazywane między procesami.
5. Zmienne `Ki` zapewniają wzajemne wykluczanie.
6. Zmienna `czyja_kolej` służy do przekazywania prawa nalegania.

Alg. Dekkera (poprawny) – opis 2/2

6. Każdy proces sprawdza, czy teraz jest jego kolej na naleganie. Jeśli nie, to przywraca początkową wartość zmiennej (`K1` na `1` dla `P1`, `K2` na `2` dla `P2`), po czym cierpliwie czeka na swoją kolej.
7. Jeżeli proces wejdzie do protokołu wstępnego, to po pewnym czasie wejdzie do sekcji krytycznej.
8. Żaden proces nie może być zagłodzony i przy braku współzawodnictwa proces może natychmiast wejść do swojej sekcji krytycznej.
9. Protokół wstępny: linie 7-15, 25-33
10. Protokół końcowy: linie 17-18, 35-36

Wzajemne wykluczanie – alg. Petersona

```
1.  K1, K2 : Integer := 1;
2.  czyja_kolej: Integer := 1;

3.  process P1 is
4.  begin
5.    loop
6.      Sekcja_lokalna_1;
7.      K1 := 0;
8.      czyja_kolej := 2;
9.      while K2 = 0 AND
              czyja_kolej = 2
          // czekaj
10.     end while;
11.     Sekcja_krytyczna_1;
12.     K1 := 1;
13.   end loop;
14. end P1;
```

```
15. process P2 is
16. begin
17.   loop
18.     Sekcja_lokalna_2;
19.     K2 := 0;
20.     czyja_kolej := 1;
21.     while K1 = 0 AND
              czyja_kolej = 1
          // czekaj
22.     end while;
23.     Sekcja_krytyczna_2;
24.     K2 := 1;
25.   end loop;
26. end P2;
```

Alg. Petersona – opis

1. Algorytm Petersona podobny jest do alg. Dekkera.
   - Zmienne `Ki` zapewniają wzajemne wykluczanie.
   - Zmienna `czyja_kolej` służy do przekazywania prawa nalegania – teraz w protokole wejściowym.
2. Każdy proces w skończonym czasie wejdzie do sekcji krytycznej.
   - Żaden proces nie może być zagłodzony
     - przy braku współzawodnictwa – proces może natychmiast wejść do swojej sekcji krytycznej (warunek pętli == FALSE);
     - przy rywalizacji – tylko dla jednego procesu warunek pętli nie będzie spełniony (z uwagi na zmienną czyja_kolej) i przy kolejnej rywalizacji sytuacja się odwróci.
3. Protokół wstępny: odpowiednio linie 7-10, 19-22
4. Protokół końcowy: odpowiednio linie 12, 24

Wzajemne wykluczanie – alg. piekarniany (Lamporta)

```
1.  wybieranie: array [1..N] of Boolean := false;
2.  numerek: array [1..N] of Integer := 0;

3.  process P(p : Integer) is
4.  begin
5.    loop
6.      Sekcja_lokalna_p;
7.      wybieranie[p] := true;
8.      numerek[p] := MAX(numerek[1] .. numerek[N]) + 1;
9.      wybieranie[p] := false;
10.     for j := 1 to N do
11.       while wybieranie[j] do
            // czekaj
12.       end while;
13.       while numerek[j] /= 0 AND
                (numerek[j], j) < (numerek[p], p) do
            // czekaj
14.       end while;
15.     end for;
16.     sekcja_krytyczna_p;
17.     numerek[p] := 0;
18.   end loop;
19. end P;
```

Wzajemne wykluczanie – CompareExchange

```
1.  // 1 - dostęp, 0 - brak dostępu
2.  acc : Integer := 1;
3.  process P_1 is
4.  begin
5.    loop
6.      Sekcja_lokalna_1;
7.      while true
8.        if CompExch(acc, 1, 0)
9.          break;
10.       end if;
11.     end while;
12.     Sekcja_krytyczna_1;
13.     acc := 1;
14.   end loop;
15. end P_1;
```

```
16. process P_n is
17. begin
18.   loop
19.     Sekcja_lokalna_n;
20.     while true
21.       if CompExch(acc, 1, 0)
22.         break;
23.       end if;
24.     end while;
25.     Sekcja_krytyczna_n;
26.     acc := 1;
27.   end loop;
28. end P_n;
```

Podsumowanie – zapowiedź następnych spotkań

- Oprócz omówionych algorytmów (Dekkera, Petersona, Lamporta) znane są jeszcze inne algorytmy:
  - Hymana
  - Dijkstry;
- W praktyce stosuje się wysokopoziomowe mechanizmy synchronizacji:
  - semafory
  - regiony krytyczne
  - monitory
- oraz mechanizmy komunikacji
  - spotkania symetryczne i asymetryczne
  - przestrzenie krotek
  - potoki, komunikaty i kanały;