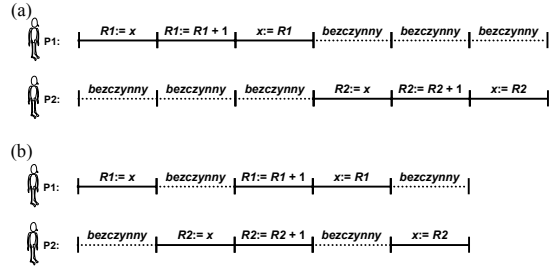


Synchronizacja cz. I

Sekwencja wykonania współbieżnych procesów

Rozważmy możliwe sekwencje wykonywania takich współbieżnych procesów.



(c) Zakład Systemów Informatycznych

Slajd 5

Problem wzajemnego wykluczania ⁽¹⁾

Rozważamy system, w którym współbieżnie wykonywane są procesy od 1 do n .

Zakładamy, że nie są znane względne prędkości wykonywania tych procesów, tzn. że liczba instrukcji wykonywanych przez poszczególne procesory w jednostce czasu może być dowolna. Przyjmujemy ponadto, że procesy te mają dostęp do wspólnych zasobów.

Rozważmy dla przykładu dwa procesy:

P1: $x := x + 1$;

P2: $x := x + 1$;



(c) Zakład Systemów Informatycznych

Slajd 3

Sformułowanie formalne problemu wzajemnego wykluczania

Dany jest zbiór procesów sekwencyjnych komunikujących się przez wspólną pamięć. Każdy z procesów zawiera sekcję krytyczną, w której następuje dostęp do wspólnej pamięci. Procesy te są procesami cyklicznymi

Zakłada się ponadto:

1. Zapis i odczyt wspólnych danych jest operacją niepodzielną, a próba jednoczesnych zapisów lub odczytów realizowana jest sekwencyjnie w nieznanym porządku.
2. Sekcje krytyczne nie mają priorytetu.
3. Względne prędkości wykonywania procesów są nieznane.
4. Proces może zostać zawieszony poza sekcją krytyczną.
5. Procesy realizujące instrukcje poza sekcją krytyczną nie mogą uniemożliwiać innym procesom wejścia do sekcji krytycznej.
6. Procesy powinny uzyskać dostęp do sekcji krytycznej w skończonym czasie

Przy tych założeniach należy zagwarantować, że w każdej chwili czasu co najwyżej jeden proces jest w swej sekcji krytycznej.

(c) Zakład Systemów Informatycznych

Slajd 6

Problem wzajemnego wykluczania ⁽²⁾

Załóżmy w tym przypadku, że każde uaktualnienie składa się z trzech faz:

- $R := x$;
pobranie wartości zmiennej x do rejestru wewnętrznego procesora
- $R := R + 1$
inkrementacja zawartości rejestru wewnętrznego procesora
- $x := R$
zapisanie zawartości rejestru do zmiennej x

(c) Zakład Systemów Informatycznych

Slajd 4

Rozwiązanie programowe problemu wzajemnego wykluczania ⁽¹⁾

```
1. program VERSIONONE;
2. var processNumber: INTEGER;

3. procedure PROCESSEONE;
4. begin
5.   while True do
6.     begin
7.       while processNumber=2 do;
8.         criticalSectionOne;
9.         processNumber:=2;
10.        otherStuffOne;
11.      end;
12.    end;

13. procedure PROCESSTWO;
14. begin
15.   while True do
16.     begin
17.       while processNumber=1 do;
18.         criticalSectionTwo;
19.         processNumber:=1;
20.         otherStuffTwo;
21.       end;
22.     end;

23. begin
24.   processNumber:= 1;
25.   parbegin
26.     PROCESSEONE;
27.     PROCESSTWO;
28.   parend;
29. end.
```



(c) Zakład Systemów Informatycznych

Slajd 7

Rozwiązanie programowe problemu wzajemnego wykluczania (2)

```

1. program VERSIONTWO;
2. var P1inside, P2inside: BOOLEAN;

3. procedure PROCESSEONE;
4. begin
5.   while True do
6.     begin
7.       while P2inside do;
8.         P1inside:=True;
9.         criticalSectionOne;
10.        P1inside:=False;
11.        otherStuffOne;
12.      end;
13.    end;

14. procedure PROCESSTWO;
15. begin
16.   while True do
17.     begin
18.       while P1inside do;
19.         P2inside:=True;
20.         criticalSectionTwo;
21.         P2inside:=False;
22.         otherStuffTwo;
23.       end;
24.     end;

25.   begin
26.     P1inside:=False;
27.     P2inside:=False;
28.     parbegin
29.       PROCESSEONE;
30.       PROCESSTWO;
31.     parend;
32.   end.

```



(c) Zakład Systemów Informatycznych

Slajd 8

Algorytm Dekkera (1)

```

1. program DEKKERALGORITHM;
2. var favoredProcess: enum (First, Second);
3. var P1WantsToEnter, P2WantsToEnter: BOOLEAN;

4. procedure PROCESSEONE;
5. begin
6.   while True do
7.     begin
8.       P1WantsToEnter:=True;
9.       while P2WantsToEnter do
10.        if favoredProcess=Second then
11.          begin
12.            P1WantsToEnter:=False;
13.            while favoredProcess=Second do;
14.              P1WantsToEnter:=True;
15.            end;
16.          criticalSectionOne;
17.          favoredProcess:=Second;
18.          P1WantsToEnter:=False;
19.          otherStuffOne;
20.        end;
21.      end;

22. procedure PROCESSTWO;
23. begin
24.   while True do
25.     begin
26.       P2WantsToEnter:= True;
27.       while P1WantsToEnter do
28.        if favoredProcess=First then
29.          begin
30.            P2WantsToEnter:=False;
31.            while favoredProcess=First do;
32.              P2WantsToEnter:= True;
33.            end;
34.          criticalSectionTwo;
35.          favoredProcess:=First;
36.          P2WantsToEnter:=False;
37.          otherStuffTwo;
38.        end;
39.      end;

40. begin
41.   P1WantsToEnter:= False;
42.   P2WantsToEnter:= False;
43.   favoredProcess:= First;
44.   parbegin
45.     PROCESSEONE;
46.     PROCESSTWO;
47.   parend;
48. end.

```



(c) Zakład Systemów Informatycznych

Slajd 11

Rozwiązanie programowe problemu wzajemnego wykluczania (3)

```

1. program VERSIONTHREE;
2. var P1WantsToEnter: BOOLEAN;
3. var P2WantsToEnter: BOOLEAN;

4. procedure PROCESSEONE;
5. begin
6.   while True do
7.     begin
8.       P1WantsToEnter:=True;
9.       while P2WantsToEnter do;
10.        criticalSectionOne;
11.        P1WantsToEnter:=False;
12.        otherStuffOne;
13.      end;
14.    end;

15. procedure PROCESSTWO;
16. begin
17.   while True do
18.     begin
19.       P2WantsToEnter:=True;
20.       while P1WantsToEnter do;
21.        criticalSectionTwo;
22.        P2WantsToEnter:=False;
23.        otherStuffTwo;
24.      end;
25.    end;

26.   begin
27.     P1WantsToEnter:=False;
28.     P2WantsToEnter:=False;
29.     parbegin
30.       PROCESSEONE;
31.       PROCESSTWO;
32.     parend;
33.   end.

```



(c) Zakład Systemów Informatycznych

Slajd 9

Algorytm Dekkera (2)

```

22. procedure PROCESSTWO;
23. begin
24.   while True do
25.     begin
26.       P2WantsToEnter:= True;
27.       while P1WantsToEnter do
28.        if favoredProcess=First then
29.          begin
30.            P2WantsToEnter:=False;
31.            while favoredProcess=First do;
32.              P2WantsToEnter:= True;
33.            end;
34.          criticalSectionTwo;
35.          favoredProcess:=First;
36.          P2WantsToEnter:=False;
37.          otherStuffTwo;
38.        end;
39.      end;

40. begin
41.   P1WantsToEnter:= False;
42.   P2WantsToEnter:= False;
43.   favoredProcess:= First;
44.   parbegin
45.     PROCESSEONE;
46.     PROCESSTWO;
47.   parend;
48. end.

```



(c) Zakład Systemów Informatycznych

Slajd 12

Rozwiązanie programowe problemu wzajemnego wykluczania (4)

```

1. program VERSIONFOUR;
2. var P1WantsToEnter: BOOLEAN;
3. var P2WantsToEnter: BOOLEAN;

4. procedure PROCESSEONE;
5. begin
6.   while True do
7.     begin
8.       P1WantsToEnter:=True;
9.       while P2WantsToEnter do
10.        begin
11.          P1WantsToEnter:=False;
12.          delay (random, freecycles);
13.          P1WantsToEnter:=True;
14.        end;
15.      criticalSectionOne;
16.      P1WantsToEnter:=False;
17.      otherStuffOne;
18.    end;

19. end;

20. procedure PROCESSTWO;
21. begin
22.   while True do
23.     begin
24.       P2WantsToEnter:=True;
25.       while P1WantsToEnter do
26.        begin
27.          P2WantsToEnter:=False;
28.          delay (random, freecycles);
29.          P2WantsToEnter:=True;
30.        end;
31.      criticalSectionTwo;
32.      P2WantsToEnter:=False;
33.      otherStuffTwo;
34.    end;

35.   begin
36.     P1WantsToEnter:=False;
37.     P2WantsToEnter:=False;
38.     parbegin
39.       PROCESSEONE;
40.       PROCESSTWO;
41.     parend;
42.   end.

```



(c) Zakład Systemów Informatycznych

Slajd 10

Algorytm Dijkstry

```

1. program DIJKSTRAALGORITHM;
2. begin
3.   shared
4.     flag[1..n]: 0..2;
5.     turn : 1..n;
6.   local
7.     test: 0..2;
8.     k, other: i, temp: 1..n;
9.   while True do
10.    begin
11.      L: flag[i]:=1;
12.      other:=turn;
13.      while other≠i do
14.        begin
15.          test:=flag[other];
16.          if test=0 then
17.            turn:=i;
18.            other:=turn;
19.          end;
20.          flag[i]:=2;

21.      for k:=1 to n do
22.        if k≠i then
23.          begin
24.            test:=flag[k];
25.            if test=2 then
26.              goto L;
27.          end;
28.      criticalSection;
29.      flag[i]:=0;
30.      reminderSection;
31.    end;
32.  end.

```



(c) Zakład Systemów Informatycznych

Slajd 13

Algorytm Petersona dla 2 procesów

```

1. program PATERSONALGORITHM;
2. begin
3.   shared
4.     flag[0..1]: BOOLEAN;
5.     turn: INTEGER;
6.   local
7.     otheri: BOOLEAN;
8.     whosei: INTEGER;
9.   while True do
10.    begin
11.      flag[i]:= True;
12.      turn:=1-i;
13.      repeat
14.        whosei:=turn;
15.        otheri:=flag[1-i];
16.      until (whosei=i or not otheri);
17.      criticalSection;
18.      flag[i]:= False;
19.      reminderSection;
20.    end;
21. end.

```



Algorytm Petersona dla n procesów

```

1. program PATERSONALGORITHM_N;
2. begin
3.   shared
4.     flag[1..n]: INTEGER;
5.     turn[1..n-1]: INTEGER;
6.   local
7.     k, l, otheri, whosei: INTEGER;
8.   while True do
9.     begin
10.      for k:=1 to n-1 do
11.        begin
12.          flag[i]:=k;
13.          turn[k]:=i;
14.          repeat
15.            whosei:=turn[k];
16.            if whosei≠i then break;
17.            for l:=1 to n do
18.              begin
19.                if l≠i then
20.                  otheri:=flag[l];
21.                  if otheri≥k then break;
22.                end;
23.              until otheri<k;

```

```

24.       end;
25.       criticalSection;
26.       flag[i]:=0;
27.       reminderSection;
28.     end;
29.   end.

```



Algorytm Lamporta dla n procesów

```

1. program LAMPORTALGORITHM;
2. begin
3.   shared
4.     choosing[1..n]: 0..1;
5.     num[1..n]: INTEGER;
6.   local
7.     testi: 0..1;
8.     k, minei: INTEGER;
9.     otheri, tempi: INTEGER;
10.  while True do
11.    begin
12.      choosing[i]:=1;
13.      for k:=1 to n do
14.        if k≠i then
15.          begin
16.            tempi:=num[k];
17.            minei:=max(minei, tempi);
18.          end;
19.      minei:= minei+1;
20.      num[i]:= minei;
21.      choosing[i]:=0;
22.      for k:=1 to n do
23.        if k≠i then
24.          begin
25.            repeat
26.              testi:=choosing[k];
27.            until testi=0;
28.            repeat
29.              otheri:=num[k];
30.            until otheri=0 or
31.              (minei, i)<(otheri, k);
32.          end;
33.        criticalSection;
34.        num[i]:=0;
35.        reminderSection;
36.      end;

```

