Programowanie współbieżne

Semafory

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

Semafor – definicja

- Semafor jest zmienną całkowitą nieujemną lub – w przypadku semaforów binarnych – zmienną typu logicznego.
- Na semaforze można wykonać dwa rodzaje atomowych operacji:
  - **P** – opuszczanie semafora (hol. passeren – *przejść*, proberen – *próbować*);
  - **V** – podnoszenie semafora (hol. vrijgeven – *zwalniać*, verhogen – *zwiększać*).
- Synchronizacja polega na:
  - blokowaniu procesu w operacji opuszczania semafora, jeśli semafor był już opuszczony (zmienna ma wartość 0);
  - wznawianiu jednego z zablokowanych wcześniej procesów w operacji podnoszenia, jeśli semafor był już opuszczony.

Rodzaje semaforów

- **Semafor binarny** – zmienna semaforowa przyjmuje tylko dwie wartości:
  - true lub 1 (stan podniesienia, otwarcia)
  - false lub 0 (stan opuszczenia, zamknięcia).
- **Semafor ogólny (zliczający)** – zmienna semaforowa przyjmuje wartości całkowite nieujemne, a jej bieżąca wartość jest zmniejszana lub zwiększana o 1 w wyniku wykonania odpowiednio operacji opuszczenia lub podniesienia semafora.

Rodzaje semaforów

- **Semafor uogólniony** – odmiana semafora ogólnego (zliczającego), w przypadku którego zmienną semaforową można zwiększać lub zmniejszać o dowolną wartość całkowitą, podaną jako argument operacji.
- **Semafor dwustronnie ograniczony** – semafor ogólny, w przypadku którego zmienna semaforowa, oprócz dolnego ograniczenia wartością 0, ma górne ograniczenie, podane przy definicji semafora.

Rodzaje semaforów z punktu widzenia sterowania procesami

- **Semafor silnie uczciwy** (***ang. strongly-fair semaphore***)
  - jeśli operacja podnoszenia jest wykonywana na semaforze nieskończenie wiele razy, to w końcu każdy oczekujący proces zakończy wykonywanie operacji opuszczania semafora;
- **Semafor słabo uczciwy** (***ang. weakly-fair semaphore***)
  - jeśli semafor stale ma wartość większą od zera, to w końcu każdy oczekujący proces zakończy wykonywanie operacji opuszczania semafora;

Rodzaje implementacji semaforów

- Semafor z aktywnym czekaniem (*ang. busy-wait semaphore*)
- Semafor ze zbiorem procesów oczekujących (*ang. blocked-set semaphore*)
- Semafor z kolejką procesów oczekujących (*ang. blocked-queue semaphore*)

Implementacja semafora ogólnego z aktywnym czekaniem (instrukcja atomowa CAS)

```
1.  procedure P(s: Integer)
2.    b: Boolean := FALSE;
3.    ss: Integer;
4.  begin
5.    while NOT b do
6.      ss := s;
7.      if ss > 0 then
8.        b = CAS(ss, s, ss - 1);
9.      end if;
10.   end while;
11. end;

12. procedure V(s: Integer)
13.   b: Boolean;
14. begin
15.   b = CAS(s, s, s + 1);
16. end;
```

```
17. procedere CAS(expVal, currVal,
        newVal: Integer): boolean
18. begin
19.     if expVal == currVal then
20.         currVal = newVal;
21.         return TRUE;
22.     else
23.         return FALSE;
24.     end if;
25. end;
```

Implementacja semafora ogólnego z kolejką procesów (1/2)

```
1. type Semaphore = record
2.    wartość: Integer;
3.    L: list of Proces;
4. end;

5. procedure P(s: Semaphore)
6. begin
7.    s.wartość := s.wartość – 1;
8.    if s.wartość < 0 then
9.       dołącz dany proces do kolejki s.L
10.      zmień stan danego procesu na „oczekujący”
11.   end if;
12.end;
```

Implementacja semafora ogólnego z kolejką procesów (2/2)

```
1. procedure V(s: Semaphore)
2. begin
3.    s.wartość := s.wartość + 1;
4.    if s.wartość <= 0 then
5.       wybierz i usuń jakiś/kolejny proces z kolejki s.L
6.       zmień stan wybranego procesu na „gotowy”
7.    end if;
8. end;
```

Implementacja semafora binarnego z kolejką procesów

```
1. procedure V(s: Bin_Semaphore)
2. begin
3.    if s.wartość < 1 then
4.       s.wartość := s.wartość + 1;
5.       if s.wartość <= 0 then
6.          wybierz i usuń jakiś/kolejny proces z kolejki s.L
7.          zmień stan wybranego procesu na „gotowy”
8.       end if;
9.    end if;
10.end;
```

Problem wzajemnego wykluczania

```
1. S : Semaphore := 1;

2. procedure Proc(i : Integer)
3. begin
4.    loop
5.       Sekcja_lokalna;
6.       P(S); // sekcja wejściowa (protokół wstępny)
7.       Sekcja_krytyczna;
8.       V(S); // sekcja wyjściowa (protokół końcowy)
9.    end loop;
10.end Proc;
```

Wykorzystanie semaforów

- Klasyczne problemy synchronizacji:
  - Problem producenta i konsumenta – problem ograniczonego buforowania w komunikacji międzyprocesowej;
  - Problem czytelników i pisarzy – problem synchronizacji dostępu do zasobu w trybie współdzielonym i wyłącznym;
  - Problem pięciu filozofów – problem jednoczesnego dostępu do dwóch zasobów (ryzyko głodzenia i zakleszczenia);
  - Problem śpiących fryzjerów – problem synchronizacji w interakcji klient-serwer przy ograniczonym kolejkowaniu;
  - Problem palaczy – problem synchronizacji wyłącznego dostępu do ograniczonej puli różnych zasobów.

Problem producenta i konsumenta – opis

- Mamy dwa typy procesów:
  - producent – produkuje jednostki określonego produktu i umieszcza je w buforze o ograniczonym rozmiarze;
  - konsument – pobiera jednostki produktu z bufora i konsumuje je.
- Z punktu widzenia producenta problem synchronizacji polega na tym, że nie może on umieścić kolejnej jednostki, jeśli bufor jest pełny.
- Z punktu widzenia konsumenta problem synchronizacji polega na tym, że nie powinien on mieć dostępu do bufora, jeśli nie ma tam żadnego elementu do pobrania.

Problem 1 producenta i 1 konsumenta (1/2)

```
1. const N: Integer := rozmiar_bufora;
2. buf: array [0..N - 1] of ElemT;
3. wolne: Semaphore := N;
4. zajęte: Semaphore := 0;
```

Problem 1 producenta i 1 konsumenta (2/2)

```
5.  procedure Producent()
6.     i: Integer := 0;
7.     elem: ElemT;
8.  begin
9.     loop
10.       produkuj(elem);
11.       P(wolne);
12.       buf[i] := elem;
13.       V(zajęte);
14.       i := (i + 1) mod N;
15.    end loop;
16. end;
```

```
17. procedure Konsument()
18.    i: Integer := 0;
19.    elem: ElemT;
20. begin
21.    loop
22.       P(zajęte);
23.       elem := buf[i];
24.       V(wolne);
25.       i := (i + 1) mod N;
26.       konsumuj(elem);
27.    end loop;
28. end;
```

Problem wielu producentów i konsumentów (1/2)

```
1. const N: Integer := rozmiar_bufora;
2. buf: array [0..N - 1] of ElemT;
3. wolne: Semaphore := N;
4. zajęte: Semaphore := 0;
5. j: Integer := 1;
6. k: Integer := 1;
7. chroń_j: BinarySemaphore := true;
8. chroń_k: BinarySemaphore := true;
```

Problem wielu producentów i konsumentów (2/2)

```
9.  procedure Producent(i: Integer)
10.   elem: ElemT;
11. begin
12.   loop
13.     produkuj(elem);
14.     P(wolne);
15.     P(chroń_j);
16.     buf[j] := elem;
17.     V(zajęte);
18.     j := (j + 1) mod N;
19.     V(chroń_j);
20.   end loop;
21. end;
```

```
22. procedure Konsument(i: Integer)
23.   elem: ElemT;
24. begin
25.   loop
26.     P(zajęte);
27.     P(chroń_k);
28.     elem := buf[k];
29.     V(wolne);
30.     k := (k + 1) mod N;
31.     V(chroń_k);
32.     konsumuj(elem);
33.   end loop;
34. end;
```

Problem czytelników i pisarzy – opis

- Dwa rodzaje użytkowników – czytelnicy i pisarze – korzystają ze wspólnego zasobu – czytelni.
- Czytelnicy korzystają z czytelni w trybie współdzielonym, tzn. w czytelni może przebywać w tym samym czasie wielu czytelników.
- Pisarze korzystają z czytelni w trybie wyłącznym, tzn. w czasie, gdy w czytelni przebywa pisarz, nie może z niej korzystać innym użytkownik (ani czytelnik, ani inny pisarz).
- Synchronizacja polega na blokowaniu użytkowników przy wejściu do czytelni, gdy wymaga tego tryb dostępu.

Problem czytelników i pisarzy – faworyzacja czytelników (1/3)

```
1. l_czyt : Integer := 0;
2. mutex_c : BinarySemaphore := true;
3. mutex_p : BinarySemaphore := true;
```

Problem czytelników i pisarzy – faworyzacja czytelników (2/3)

```
4.  procedure Czytelnik()
5.  begin
6.    loop
7.      własne_sprawy;
8.      P(mutex_c);
9.      l_czyt := l_czyt + 1;
10.     if l_czyt = 1 then
11.       P(mutex_p);
12.     V(mutex_c);
13.     czytanie;
14.     P(mutex_c);
15.     l_czyt := l_czyt - 1;
16.     if l_czyt = 0 then
17.       V(mutex_p);
18.     V(mutex_c);
19.   end loop;
20. end;
```

```
21. procedure Pisarz()
22. begin
23.   loop
24.     własne_sprawy;
25.     P(mutex_p);
26.     pisanie;
27.     V(mutex_p);
28.   end loop;
29. end;
```

Problem czytelników i pisarzy – rozwiązanie poprawne (1/3)

```
1. cp : Integer := 0; // liczba czytelników w poczekalni
2. cc : Integer := 0; // liczba czytelników w czytelni
3. pp : Integer := 0; // liczba pisarzy w poczekalni
4. pc : Integer := 0; // liczba pisarzy w czytelni
5. czyt : Semaphore := 0;
6. pis : Semaphore := 0;
7. chroń : BinarySemaphore := true;
```

Problem czytelników i pisarzy – rozwiązanie poprawne (2/3)

```
8.  procedure Czytelnik(i: Integer)
9.  begin
10.   loop
11.     własne_sprawy;
12.     P(chroń);
13.     if pp + pc = 0 then
14.       cc := cc + 1;
15.       V(czyt);
16.     else
17.       cp := cp + 1;
18.     end if;
19.     V(chroń);
20.     P(czyt);
21.     czytanie;
```

```
22.     P(chroń);
23.     cc := cc - 1;
24.     if cc = 0 then
25.       if pp > 0 then
26.         pc := 1;
27.         pp := pp - 1;
28.         V(pis);
29.       end if;
30.     end if;
31.     V(chroń);
32.   end loop;
33. end;
```

Problem czytelników i pisarzy – rozwiązanie poprawne (3/3)

```
34. procedure Pisarz(i: Integer)
35. begin
36.   loop
37.     własne_sprawy;
38.     P(chroń);
39.     if cc + pc = 0 then
40.       pc := 1;
41.       V(pis);
42.     else
43.       pp := pp + 1;
44.     end if;
45.     V(chroń);
46.     P(pis);
47.     pisanie;
```

```
48.     P(chroń);
49.     pc := 0;
50.     if cp > 0 then
51.       while cp > 0 do
52.         cc := cc + 1;
53.         cp := cp - 1;
54.         V(czyt);
55.       end while;
56.     else if pp > 0 then
57.       pc := 1;
58.       pp := pp - 1;
59.       V(pis);
60.     end if;
61.     V(chroń);
62.   end loop;
63. end;
```

Problem czytelników i pisarzy – rozwiązanie poprawne dla ustalonej liczby czytelników

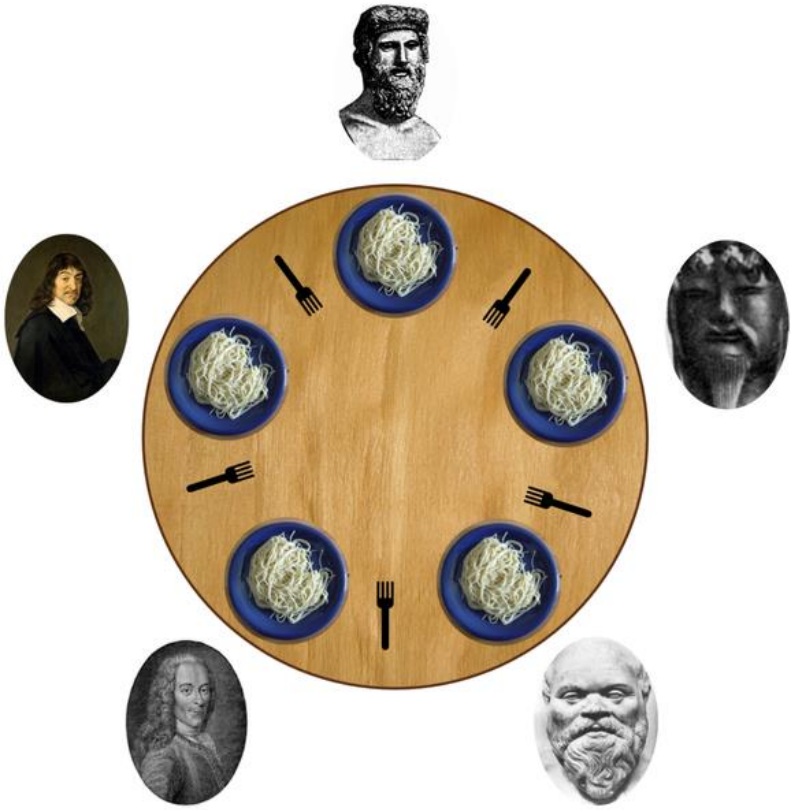
```
1.  wolne : Semaphore := M;
2.  pis : BinarySemaphore := true;

3.  procedure Czytelnik(i: Integer)
9.  begin
10.   loop
11.     własne_sprawy;
12.     P(wolne);
13.     czytanie;
14.     V(wolne);
15.   end loop;
16. end;
```

```
17. procedure Pisarz(i: Integer)
18.   j: Integer;
19. begin
20.   loop
21.     własne_sprawy;
22.     P(pis);
23.     for j := 1 TO M do
24.       P(wolne);
25.     end for;
26.     pisanie;
27.     for j := 1 TO M do
28.       V(wolne);
29.     end for;
30.     V(pis);
31.   end loop;
32. end;
```

Problem pięciu filozofów – opis

- Przy okrągłym stole siedzi pięciu filozofów, którzy na przemian myślą (filozofują) i jedzą makaron ze swoich misek.
- Żeby coś zjeść, filozof musi zdobyć dwa widelce, z których każdy współdzieli ze swoim sąsiadem.
- Widelec dostępny jest w trybie wyłącznym – może być używany w danej chwili tylko przez jednego filozofa.
- Należy zsynchronizować filozofów tak, aby każdy mógł się w końcu najeść przy zachowaniu reguł dostępu do widelców oraz przy możliwie dużej przepustowości w spożywaniu posiłków.



zródło: Benjamin D. Esham / Wikimedia Commons

Problem pięciu filozofów – rozwiązanie z blokadą

```
1. widelec: array [0..4] of BinarySemaphore := true;

2. procedure Filozof(i: Integer)
3. begin
4.    loop
5.       myślenie();
6.       P(widelec[i]);
7.       P(widelec[(i + 1) mod 5]);
8.       jedzenie();
9.       V(widelec[i]);
10.      V(widelec[(i + 1) mod 5]);
11.   end loop;
12.end;
```

Problem pięciu filozofów – rozwiązanie poprawne

```
1.  dopuść: Semaphore := 4;
2.  widelec: array [0..4] of BinarySemaphore := true;

3.  procedure Filozof(i: Integer)
4.  begin
5.     loop
6.        myślenie();
7.        P(dopuść);
8.        P(widelec[i]);
9.        P(widelec[(i + 1) mod 5]);
10.       jedzenie();
11.       V(widelec[i]);
12.       V(widelec[(i + 1) mod 5]);
13.       V(dopuść);
14.    end loop;
15. end;
```

Problem śpiących fryzjerów/golibrodów – opis

- W salonie fryzjerskim jest *n* foteli obsługiwanych przez fryzjerów oraz poczekalnia z liczbą miejsc *p*;
- Do salonu przychodzi klient, budzi fryzjera, po czym fryzjer znajduje wolny fotel i obsługuje klienta;
- Jeśli nie ma wolnego fotela, klient zajmuje jedno z wolnych miejsc w poczekalni;
- Jeśli nie ma miejsca w poczekalni, to klient odchodzi;
- Problem polega na zsynchronizowaniu fryzjerów oraz klientów w taki sposób, aby jeden fryzjer w danej chwili obsługiwał jednego klienta i w tym samym czasie klient był obsługiwany przez jednego fryzjera.

Problem śpiących fryzjerów – rozwiązanie (1/3)

```
1. const poj_poczek : Integer := pojemność_poczekalni;
2. const licz_foteli: Integer := licz;
3. l_czek : Integer := 0; // liczba klientów w poczekalni
4. mutex : BinarySemaphore := true;
5. klient : Semaphore := 0;
6. fryzjer : Semaphore := 0;
7. fotel : Semaphore := licz_foteli;
```

Problem śpiących fryzjerów – rozwiązanie (2/3)

```
8.  procedure Klient(i: Integer)
9.  begin
10.   loop
11.     sprawy_własne();
12.     P(mutex);
13.     if l_czek < poj_poczek then
14.       l_czek := l_czek + 1;
15.       V(klient);
16.       V(mutex);
17.       P(fryzjer);
18.       strzyżenie();
19.     else
20.       V(mutex); // odchodzi z zakładu nieobsłużony
21.     end if;
22.   end loop;
23. end; {Klient}
```

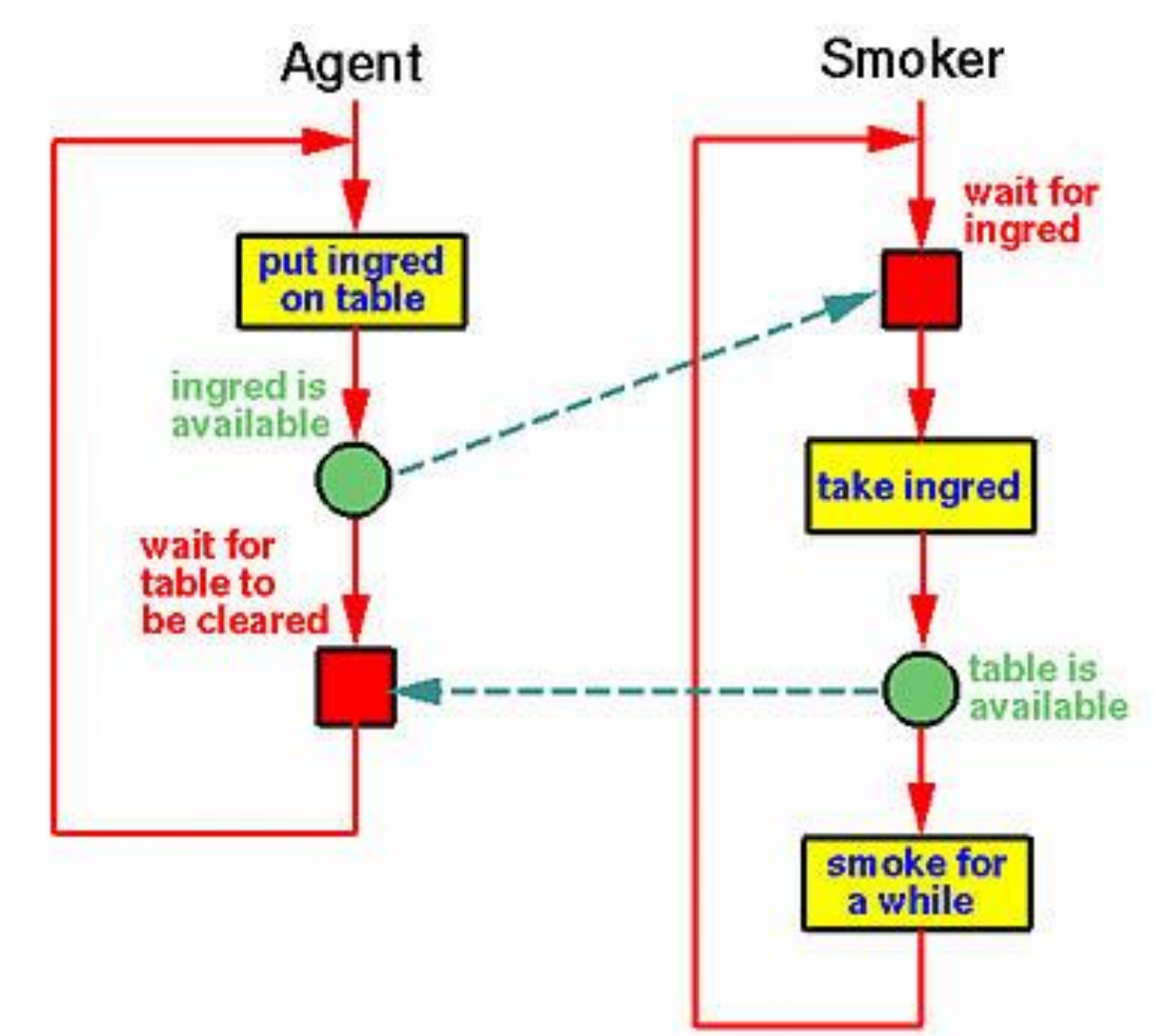
Problem śpiących fryzjerów – rozwiązanie (3/3)

```
24. procedure Fryzjer(i: Integer)
25. begin
26.   loop
27.     P(klient);
28.     P(fotel);
29.     P(mutex);
30.     l_czek := l_czek - 1;
31.     V(fryzjer);
32.     V(mutex);
33.     strzyżenie();
34.     V(fotel);
35.     sprawy_własne();
36.   end loop;
37. end; {Fryzjer}
```

Problem palaczy – opis (1/2)

- Problem został opisany w 1971r. przez S. S. Patil'a.
- Zakładamy, że palenie papierosów wymaga 3 składników: tytoniu, papieru, zapałki.
- Każdy z 3 palaczy posiada tylko jeden ze składników w nieograniczonej ilości.
- Agent posiada nieograniczoną ilość każdego ze składników.
- Agent i każdy palacz współdzielą stół.
- Agent losowo kładzie na stół dwa składniki i powiadamia palaczy potrzebujących tych dwóch składników.
- Agent kładzie nowe składniki, gdy poprzednie zostały zabrane ze stołu.
- Palacz, który został powiadomiony o potrzebnych mu składnikach, bierze je ze stołu, robi papieros i pali go przez chwilę, po czym wraca do oczekiwania na kolejną partię potrzebnych mu składników na stole.

Problem palaczy – opis (2/2)



Problem palaczy – rozwiązanie (1/4)

```
1. składniki: array [0..2] of bool := false;
2. dostęp: BinarySemaphore := true;
3. zabrane: Semaphore := 2;
4. czekaj: array [0..2] of BinarySemaphore := false;
```

Problem palaczy – rozwiązanie (2/4)

```
5.  procedure Agent()
6.    i, j: integer;
7.  begin
8.    loop
9.      losuj_składniki(i);
10.     P(zabrane);
11.     P(zabrane);
12.     P(dostęp);
13.     składniki[i] := true;
14.     składniki[(i + 1) mod 3] := true;
15.     V(dostęp);
16.     for j := 0 to 2 loop
17.       V(czekaj[j]);
18.     end for;
19.   end loop;
20. end;
```

Problem palaczy – rozwiązanie z blokadą (3/4)

```
21. procedure Palacz(nr: integer)
22.   ile1, ile2: integer := 0;
23.   nr1 : integer := (nr + 1) mod 3;
24.   nr2 : integer := (nr + 2) mod 3;
25. begin
26.   loop
27.     while ile1 + ile2 < 2 loop
28.       P(dostęp);
29.       if składniki[nr1]
             AND ile1 = 0 then
30.         składniki[nr1] := false;
31.         ile1 := 1;
32.         V(zabrane);
33.       end if;
34.       if składniki[nr2]
             AND ile2 = 0 then
35.         składniki[nr2] := false;
36.         ile2 := 1;
37.         V(zabrane);
38.       end if;
```

```
40.         V(dostęp);
41.       end if;
42.       if ile1 + ile2 < 2 then
43.         P(czekaj[nr]);
44.       end if;
45.     end while;
46.     palenie();
47.     ile1 := 0;
48.     ile2 := 0;
49.   end loop;
50. end;
```

Problem palaczy – rozwiązanie poprawne (4/4)

```
21. ile: array [0..2, 0..2] of integer := 0;

22. procedure Palacz(nr: integer)
23.   nr1 : integer := (nr + 1) mod 3;
24.   nr2 : integer := (nr + 2) mod 3;
25. begin
26.   ile[nr][nr] = 1;
27.   loop
28.     while ile[nr][nr1]
29.           + ile[nr][nr2] < 2 do
30.       P(dostęp);
31.       if składniki[nr1]
32.          AND ile[nr][nr1] = 0 then
33.         if ile[nr][nr1]
34.            + ile[nr][nr2] > 0
35.            OR nie_zablokuje(nr1) then
36.           składniki[nr1] := false;
37.           ile[nr][nr1] := 1;
38.           V(zabrane);
39.         end if;
40.       end if;
41.       if składniki[nr2]
42.          AND ile[nr][nr2] = 0 then
43.         if ile[nr][nr1]
44.            + ile[nr][nr2] > 0
45.            OR nie_zablokuje(nr2) then
46.           składniki[nr2] := false;
47.           ile[nr][nr2] := 1;
48.           V(zabrane);
49.         end if;
50.       end if;
51.       V(dostęp);
52.       if ile[nr][nr1]
53.          + ile[nr][nr2] < 2 then
54.         P(czekaj[nr]);
55.       end if;
56.     end while;
57.     palenie();
58.     ile1 := 0;
59.     ile2 := 0;
60.   end loop;
61. end;
```

Problem palaczy – rozwiązanie (2/4)

```
62. procedure nie_zablokuje(nr_test : integer) : boolean
63.   i, nr1, nr2 : integer;
64. begin
65.   for i := 0 to 2 loop
66.     nr1 := (i + 1) mod 3;
67.     nr2 := (i + 2) mod 3;
68.     if ile[i][nr1] + ile[i][nr2] == 1
69.        AND ile[i][nr_test] = 0 then
70.       return false;
71.     end if;
72.   end for;
73.   return true;
74. end;
```