Przykłady rozwiązań problemów

- Semafor wymyślony został po to aby umożliwić realizację wzajemnego wykluczania

Rozwiązanie problemu wzajemnego wykluczania dowolnej liczby procesów:

```
const     N = ?                 {ilość procesów}
var       S : binary semaphore  :=1;
```

process P(i : 1..N);

```
begin
    while true do begin
    własne sprawy;
        PB(S);
            sekcja krytyczna;
        VB(S);
    end
end;
```

- Semafor ma wartość 1, więc tylko 1 proces będzie mógł zakończyć PB(S)

Producenci i konsumenci

- Bufor N - elementowy
- Dwa semafory:
  - WOLNE – w buforze są wolne pozycje;
  - PEŁNE – są zapełnione elementy;

```
const   N = ?          {rozmiar bufora}
var     WOLNE :   semaphore   := N;
        PEŁNE :   semaphore   := 0;
        bufor: array [1..N] of porcja;
```

process PRODUCENT;

```
var p : porcja;
j : 1..N        :=      1;

begin
    while true do begin
                produkuj(p);
                P(WOLNE);
                    bufor [j] := p;
                    j := j mod N +1;
                V(PEŁNE);
                end
end;
```

process KONSUMENT;

```
var p: porcja;
k: 1..N         :=      1;

begin
    while true do begin
                P(PEŁNE);
                    p := bufor [k];
                    k = k mod N + 1;
                V(WOLNE);
                konsumuj(p);
                end
end;
```

- Rozwiązanie poprawne w przypadku jednego producenta i jednego konsumenta
- Operacje na semaforach są tak wykonywane, że zmienna j nigdy nie ma tej samej wartości co zmienna k
- Równoczesne wpisywanie do bufora i pobieranie z niego
- Tylko dla jednego producent i jednego konsumenta

Wielu producentów i wielu konsumentów

- Wykluczanie producentów względem siebie
- Ochrona zmiennej j
- Wykluczanie konsumentów względem siebie
- Ochrona zmiennej k
- Dostęp do tablicy przez zmienne globalne

```
const  P = ? {liczba producentów}
       K = ? {liczba konsumentów}
       N = ? {rozmiar bufora}
```

```
var  WOLNE : semaphore := N;
     PEŁNE : semaphore := 0;
     bufor : array [1..N] of porcja;
     j : integer :=1;
     k : integer :=1;
     Chroń_j : binary semaphore :=1;
     Chroń_k: binary semaphore :=1;
```

```
process PRODUCENT(i:1..P);
var p: porcja;
begin
    while true do begin
        produkuj(p);
        P(WOLNE);
            PB(Chroń_j);
                bufor[j] := p;
                j := j mod N +1;
            VB(Chroń_j);
        V(PEŁNE);
    end
end
```

```
process KONSUMENT(i:1..K);
var p : porcja;
begin
    while true do begin
        P(PEŁNE);
            PB(Chroń_k);
                p := bufor[k];
                k := k mod N + 1;
            VB(CHROŃ_K);
        V(WOLNE);
        konsumuj(p);
    end
end;
```

- Tylko jeden producent może produkować i jeden konsument może pobierać w danej chwili czasu
- Sekcją krytyczną dla producentów jest dostęp do zmiennej j
- Sekcją krytyczną dla konsumentów jest dostęp do zmiennej k

Czytelnicy i pisarze

- Liczba czytelników M jest znana lub w czytelni może przebywać co najwyżej M czytelników

- WOLNE – liczba wolnych miejsc w czytelni
- W – wzajemne wykluczanie pisarzy

```
const M = ?  {liczba czytelników}
      P = ? {liczba pisarzy}

var   WOLNE : semaphore := M          {ilość miejsc}
      W : binary semaphore := 1       {do wykluczania pisarzy}
```

process CZYTELNIK (i : 1..M);

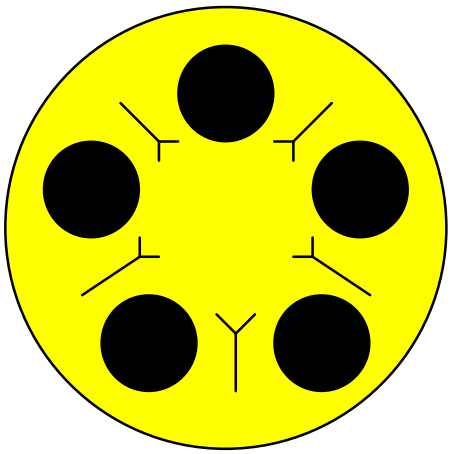
```
begin
    while true do begin
        własne sprawy;
        P(WOLNE);
          czytanie;
        V(WOLNE);
    end
end
```

process PISARZ (i : 1..P);

```
var j : integer;
begin
    while true do begin
        własne sprawy;
        PB(W);
            for j:=1 to M P(WOLNE);
              pisanie;
            for j:=1 to M V(WOLNE);
        VB(W);
    end
end;
```

- Podniesienie semafora WOLNE powoduje pojawienie się wolnego miejsca;
- Czytelnik może wejść do czytelni
- Pisarz musi zająć wszystkie miejsca w czytelni, robi to stopniowo;

Pięciu filozofów



- Rozwiązanie z możliwością blokady
- Semafor binarny WIDELEC[i];

var

WIDELEC: array[0..4] of binary semaphore := {1,1,1,1,1};

process FILOZOF(i : 0..4);

begin

while true do begin

myślenie;

PB(WIDELEC[i]);

PB(WIDELEC[(i+1) mod 5];

jedzenie;

VB(WIDELEC[i]);

VB(WIDELEC[(i+1) mod 5]);

end

end;

Rozwiązanie poprawne

- LOKAJ semafor dopuszcza 4 filozofów jednocześnie;

```
var  WIDELEC: array[0..4] of binary semaphore:={1,1,1,1,1};
     LOKAJ: semaphore :=4;
```

```
proces FILOZOF( i : 0..4 );

begin
        while true do begin
                myślenie;
                P(LOKAJ);
                        PB(WIDELEC[i]);
                        PB(WIDELEC[(i+1) mod 5];
                            jedzenie;
                        VB(WIDELEC[i]);
                        VB(WIDELEC[(i+1) mod 5]);
                V(LOKAJ);
        end
end;
```

- bez możliwości blokady i zagłodzenia;
- tylko czterech filozofów ma jednocześnie dostęp do widelców
- rozwiązanie spełnia kryteria poprawności