Programowanie Współbieżne

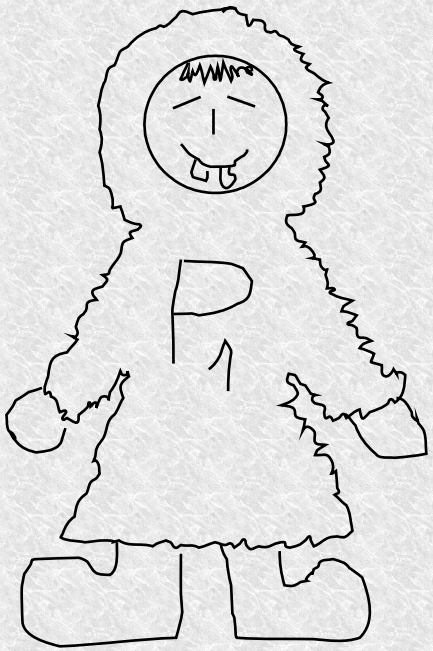
Wzajemne wykluczanie

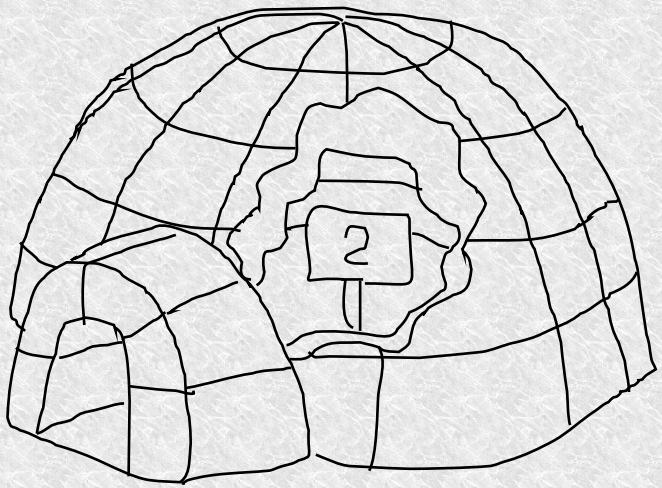
Sformułowanie problemu

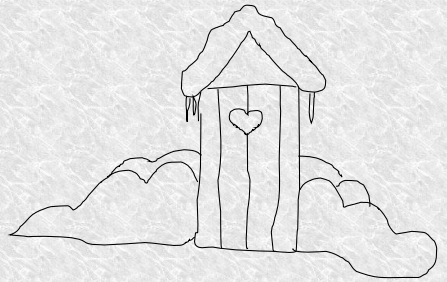
Każdy z dwóch procesów P1 i P2 wykonuje w nieskończonej pętli program składający się z dwóch części: Strefy krytycznej (odpowiednio kryt1 i kryt2) oraz strefy lokalnej (lok1 i lok2), stanowiącej resztę programu. Wykonanie kryt1 i kryt2 nie może odbywać się równocześnie.

1. Próba

Przykład zaczerpnięty z książki M. Ben-Ari „Podstawy Programowania Współbieżnego”







1. Próba

- Eskimos P1” będącym procesem 1 oraz „P2” – proces drugi.
- protokołowe igloo zawierające tabliczkę „czyja_kolej” z wąskim wejściem (arbiter pamięci) tak że w danym momencie w igloo mieści się tylko jeden Eskimos.
- mamy też strefę krytyczną do której może się udać tylko jeden proces (załóżmy że jest tam P1)

1. Próba

- Proces P2 czekający na swoją kolej bezczynnie chodzi dookoła igloo
- i co jakiś czas zagląda do środka czy przypadkiem nie jego kolej (aktywne sprawdzanie)
- nagle wraca proces P1 i zapisuje na tabliczce 2
- gdy proces P2 zagląda widzi że na tabliczce pojawił się jego numer

1. Próba

```
program pierwsza_proba;
var czyja_kolej : integer;
procedure p1;
begin
    repeat
        while czyja_kolej=2 do {nic}; {aktywne czekanie}
        kryt1; { wejście do sekcji krytycznej1}
        czyja_kolej:=2;
        lok2;  { wejście do swojej sekcji lokalnej
    until false {bez końca}
end;


procedure p2;
begin
    repeat
        while czyja_kolej=1 do {nic}; {aktywne czekanie}
        kryt2; { wejście do sekcji krytycznej1}
        czyja_kolej:=1;
        lok2;  { wejście do swojej sekcji lokalnej
    until false {bez końca}
end;

BEGIN
    czyja_kolej=1;
    cobegin
        p1;p2;
    coend;
END
```

1. Próba

Program spełnia wymagania wzajemnego wykluczania bo:

- proces P1 wchodzi do strefy krytycznej gdy czyja_kolej=1, to jest zmienna gdzieś w pamięci dzielonej
- wartość czyja_kolej pozostaje nie zmieniona aż do momentu gdy proces opuści sekcje krytyczną, do tego momentu drugi proces nie zostanie dopuszczony do sekcji krytycznej

1. Próba

Blokada nie możliwa bo:

- czyja_kolej przyjmuje wartości 1 lub 2 więc zawsze jeden proces będzie w stanie kontynuować swoją pracę, nie jest możliwe by oba procesy utknęły w pętli *while*
- ponieważ wykonanie każdej instrukcji w programie zabierze skończoną ilość czasu to w skończonym czasie dojdzie do zamiany miejscami nie dojdzie do zagłodzenia

1. Próba

Wady:

- Procesy nie są luźno połączone
  - Gdy jeden proces 100 razy na dzień a drugi tylko 1, równamy to wolniejszego.
  - Gdy jeden proces zginie (nawet poza strefą krytyczną) to ten drugi automatycznie jest blokowany.

2. Próba

- mamy 2 oddzielne igloo (zmienne globalne) C1 i C2
- jeżeli proces P1 jest w swojej strefie krytycznej to C1 = 0
- jeżeli jest poza strefą krytyczną to C1=1
- P2 sprawdza czy C1 = 1 jak tak to sobie idzie do strefy krytycznej a sam u siebie zmienia na C2 = 0

2. Próba

```
Program druga_proba;
var c1,c2:integer;
procedure p1;
begin
    repeat
        while c2=0 do;
        c1:=0;
        kryt1;
        c1:=1;
        lok1;
    until false
end;

procedure p2;
begin
    repeat
        while c1=0 do;
        c2:=0;
        kryt2;
        c2:=1;
        lok2;
    until false
end;
```

Pomiędzy *while* i przypisaniem może upłynąć dowolnie długa chwila. Jest możliwość że oba procesy znajdą się w sekcji krytycznej!

```
BEGIN
    c1:=1;
    c2:=1;
    cobegin;
        p1;p2;
    coend;
END.
```

3. Próba

```
Program druga_proba;
var c1,c2:integer;
procedure p1;
begin
    repeat
        c1:=0;
        while c2=0 do;
        kryt1;
        c1:=1;
        lok1;
    until false
end;

procedure p2;
begin
    repeat
        c2:=0;
        while c1=0 do;
        kryt2;
        c2:=1;
        lok2;
    until false
end;
```

Pomiędzy przypisaniem i *while* może upłynąć dowolnie długa chwila. Jest możliwość że oba procesy się zablokują!

```
BEGIN
    c1:=1;
    c2:=1;
    cobegin;
        p1;p2;
    coend;
END.
```

4. Próba

- Wprowadzamy rozwiązanie że proces chwilowo rezygnuje z chęci wejścia do strefy aby dać szanse drugiemu procesowi.
- P1 zapisuje C1:=0 jeżeli w C2 też jest 0 to wraca i wymazuje C1 dając C1=1

4. Próba

```
program czwarta_proba;
var c1, c2:integer;
procedure p1;
begin
   repeat
      c1:=0;
      while c2 = 0 do
         begin
            c1:=1;
            {przez kilka chwil
nic nie rob}
            c1:=0;
         end;
      kryt1;
      c1:=1;
      lok1;
   until false;
end;
```

```
procedure p2;
begin
   repeat
      c2:=0;
      while c1 = 0 do
         begin
c2:=1;
{ przez kilka chwil nic nie
rob}
c2:=0;
         end;
      kryt2;
      c2:=1;
      lok2;
   until false;
end;

BEGIN
   c1:=1;
   c2:=2;
   cobegin;
      p1;p2;
```

4. Próba

Rozwiązanie prawie idealne, jednak istnieje taki nieszczęśliwy przeplot, że będą sobie ustępować w nieskończoność.

Algorytm Dekkera

Połączenie 1 i 4 rozwiązania

- w pierwszym była możliwość utraty klucza i czekanie na wolniejszy proces
- w czwartym możliwość bezustannego wymieniania się
- Dodane Igloo rozjemcze „czyja_kolej” ale używane tylko wtedy gdy w obu C1 i C2 jest 0.

Algorytm Dekkera

```
Program Dekker;
var czyja_kolej:integer;
   C1,C2:integer;
Procedure p1;
begin
   repeat
      c1:=0;
      while c2=0 do
         if czyja_kolej=2 then
            begin
               c1:=1;
               while czyja_kolej=2 do;
               c1:=0;
            end;
      kryt1;
      czyja_kolej:=2;
      c1:=1;
      lok1;
   until false
end;
```

Algorytm Dekkera

- gdy p1 zapisze c1=0 a następnie stwierdzi że p2 też zapisał c2=0 to idą na konsultacje do rozjemcy
- jeżeli rozjemca ma 1 to idzie p1 a p2 musi ustąpić,
- gdy p1 skończy to nie tylko w c1 pisze 1 ale i w czyja_kolej pisze 2.