Programowanie Współbieżne

Monitory

Monitory

- Monitory są połączeniem dwóch idei.
  - Monitor jako Wielki Brat. Stara koncepcja, gdzie system operacyjny był jednym programem i pełnił funkcję nadzorcy pośredniczącego między poszczególnymi procesami.
    - Jedynie Monitor ma dostęp do pewnych obszarów pamięci
    - Jedynie Monitor może wykonać pewne funkcje np. WE/WY
    - Jeżeli proces 1 chce przekazać procesowi 2 to przekazuje to przez Monitor, albo pyta o możliwość bezpośredniej transmisji

Monitory

- strukturalizacji danych
  - Pascal był pierwszym językiem zaprojektowanym z myślą o strukturalizacji danych poprzez mechanizm określania typów.
  - Można określić że operacje mogą być wykonywane np. tylko z określonymi typami.
  - Zabezpieczenie danych przed łączeniem ich kiedy nie były do tego przeznaczone.
  - Określenie typów za pomocą operacji. Pomysł zastosowany w języku Simula 67 nosił nazwę **klasy**. Był to zbiór deklaracji danych wraz ze zbiorem operacji dozwolonych na nich

Monitory

- Klasy teraz są znane i używane. Słowo kluczowe ***synchronized*** w javie powoduje że klasa staje się monitorem.
- Bufor jako przykład monitora. Mamy tylko dwie operacje:
  - Włożenie elementu
  - Wyjęcie elementu

Monitory

- Definicja:
  - Definiuje się podając zbiór deklaracji zmiennych i procedur
  - Monitor ma też treść (w pascalu Begin i End) coś jak konstruktor. Wywoływana jest w chwili startu programu. Służy do nadania wartości początkowych.
  - Na zewnątrz monitor jawi się jako pakiet danych i procedur
  - Zmienne monitora są dostępne bezpośrednio tylko przez jego procedury (zmienne prywatne).
  - Komunikacja ze światem zewnętrznym tylko przez te procedury.
  - Zasięgiem zmiennych monitora jest sam monitor

Monitory – producent konsument

```
program producentkonsument;
const rozmiarbufora = 64;

monitor buforograniczony;
    niepusty,niepelny:condition;
    b:array [0..rozmiarbufora] of integer;
    in, out: integer;
    n: integer;
    procedure wloz(v:integer);
    begin
        if n = rozmiarbufora + 1 {bufor jest pelny }
            then
            {czekaj, aż nie będzie pełny}
            wait(niepelny);
        b[in]:=v;
        in: = in + 1;
        if in = rozmiarbufora + 1 then in:=0;
        n:=n+1;
        {zasygnalizuj że bufor nie jest pusty}
        signal(niepusty);
    end;
```

Monitory – producent konsument

```
    procedure pobiez(var v:integer);
    begin
        if n=0 {bufor jest pusty }
        then
            {czekaj, az nie będzie pusty}
            wait(niepusty);
        v := b[out];
        out := out + 1;
        if out = rozmiarbufora + 1 then out:=0;
        n:=n-1;
        {zasygnalizuj, że bufor nie jest pełny}
        signal(niepelny);
    end;
    begin {tresc monitora}
        in:=0;
        out:=0;
        n:=0;
    end;
{koniec monitora buforograniczony}
```

Monitory – producent konsument

```
procedure producent;
var v:integer;
begin
    repeat
        produkuj(v);
        wloz(v)
    until false;
end;

procedure konsument;
var v:integer;
begin
    repeat
        pobierz(v);
        konsumuj(v);
until false;
end;

BEGIN
    cobegin;
        producent;konsument;
    coend
END.
```

Zmienna warunkowa

- Zmienna warunkowa nie ma wartości w zwykłym tego słowa znaczeniu i nie wymaga inicjacji.
- Każdej zmiennej warunkowej jest nadawana wartość początkowa w postaci pustej kolejki procesów zatrzymanych na danym warunku.
- **wait**(c) – wywołujący proces zostaje zatrzymany i umieszczony w kolejce procesów zatrzymanych na tym warunku (zakładamy że jest to FIFO). Proces czeka na spełnienie warunku `c`.
- **signal**(c) – jeżeli kolejka nie jest pusta to reaktywujemy 1 proces z kolejki, jeżeli jest pusta to nic się nie dzieje (w semaforze zwiększaliśmy zmienną semaforową). Proces informuje że warunek `c` został spełniony.

Zmienna warunkowa

- Analogia do Igloo:
  - procesy które czekają na zajście warunku „c” są zamrażane w wielkiej zamrażarce gdzie z jednej strony wchodzą z drugiej wychodzą.
  - Wejście i wyjście zamrażarki jest w igloo, do którego najpierw procesy wchodzą.
  - Proces opuszczający igloo tuż przed wyjściem włącza rozmrażanie.
  - Zanim się nie rozmrozi proces czekający to żaden inny nie może wejść do igloo. Czyli realizowane jest:
    - wymaganie natychmiastowego wznowienia
    - Czynimy też zastrzeżenie że proces opuszczający igloo może wywołać *signal* tylko jeden raz

Symulacja semafora za pomocą Monitora

- Definicja semafora nie przewiduje kolejki FIFO ale też i nie zabrania.
- Każdy algorytm działający na semaforach ze słabszymi założeniami (brak FIFO) będzie działał przy mocniejszych założeniach.

Symulacja semafora za pomocą Monitora

```
program wzajemne_wykluczanie;
monitor symulacja_semafora;
var zajety:boolean;
    niezajety:condition;

Procedure P;
begin
    if zajety then
    wait(niezajety);
    zajety:=true
end;


Procedure V;
begin
    zajety:=false;
    signal(niezajety);
    end;
begin {monitor}
    zajety:=false;
end;
```

```
Procedure P1;
begin
    repeat
        P;
        kryt1;
        V;
        lok1;
    until false
end;

Procedure P2;
begin
    repeat
        P;
        kryt2;
        V;
        lok2;
    until false;
    end;

    BEGIN
        cobegin;
        P1;P2;
        coend;
    END.
```

Problem Czytelników i Pisarzy

- Jeżeli istnieją czekający pisarze to nowo przybyły czytelnik musi czekać na zakończenie pisania
- Jeżeli istnieją czytelnicy czekający na zakończenie pisania to mają oni pierwszeństwo przed następnymi pisarzami.

Problem Czytelników i Pisarzy

- Zastosowanie nowej procedury pierwotnej
  - `nonempty` Zwraca ***true*** wtedy i tylko wtedy gdy, kolejka związana ze zmienną warunkową jako parametr zawiera inne procesy. (jeżeli jest pisarz w kolejce lub pisze to czytelnicy czekają)
- Monitorowe wymaganie natychmiastowego reaktywowania gwarantuje że wszyscy czekający czytelnicy będą reaktywowani zanim kolejny czytelnik będzie dopuszczony do monitora

Problem Czytelników i Pisarzy

```
program czytelnicy_i_pisarze;
monitor czytajpisz;
var czytelnicy:integer;
    pisanie:boolean;
    gotowy_do_czytania:condition;
    gotowy_do_pisania:condition;

Procedure zacznij_czytanie;
begin
    if pisanie or
    nonempty(gotowy_do_pisania)
        then
        wait(gotowy_do_czytania);
    czytelnicy:czytelnicy+1;
    signal(gotowy_do_czytania);
{kaskadowe wpuszczenie wszystkich
czyt}
end;

Procedure zakoncz_czytanie;
begin
    czytelnicy:czytelnicy-1;
    if czytelnicy=0 then
        signal(gotowy_do_pisania);
end;
```

```
Procedure zacznij_pisanie;
begin
    if czytelnicy<>0 or pisanie
        then
        wait(gotowy_do_pisania);
    pisanie:=true;
end;

Procedure zakończ_pisanie;
begin
    pisanie:=false;
    if nonempty(gotowy_do_czytania)
{jeżeli są w kolejce czytelnicy to
zostaną obudzeni}
    then
        signal(gotowy_do_czytania)
    else
        signal(gotowy_do_pisania)
end;

begin
    czytelnicy:=0;
    pisanie:=false;
end;
```

Problem Czytelników i Pisarzy

```
procedure czytelnik;
begin
    repeat
        zacznij_czytanie;
        czytaj_dane;
        zakoncz_czytanie;
    until false
end;

procedure pisarz;
begin
    repeat
        zacznij_pisanie;
        pisz_dane;
        zakoncz_pisanie;
    until false
end;

BEGIN
    cobegin;
        czytelnik;pisarz;czytelnik;czytelnik;czytelnik;pisarz;...
    coend;
END.
```

Symulacja Monitorów za pomocą Semaforów

- Wzajemne wykluczanie procedur monitorowych można łatwo osiągnąć używając semafora binarnego **s** na początku =1
- Procedury monitorowe zaczynają się od *wait*(s) i kończą *signal*(s)
- Dla każdego warunku ***war*** potrzebujemy ***liczwar*** aby zapamiętać liczbę procesów czekających na ten warunek. Początkowo *liczwar*=0;
- Używając binarnego semafora ***semwar*** aby wstrzymać takie procesy, na początku =0;

Symulacja Monitorów za pomocą Semaforów

- Każdy proces wykonujący procedurę monitorową wait zawiesza swoje działanie
- **wait(war)** =
  - **liczwar:=liczwar+1;**
  - **signal(s);**
  - **wait(semwar);**
  - **liczwar:=liczwar-1;**
- **signal(war)** =
  - **If liczwar > 0 then signal (semwar) else signal(s);**

zwolnienie dostępu do monitora jest po to by umożliwić wejście innym.

Realizacja założenia, że procesy czekające na warunek, mają pierwszeństwo przed procesami, które zamierzają dopiero wejść do monitora.

Symulacja Monitorów za pomocą Semaforów

```
Program producent_konsument;
const rozmiar_bufora=64;
var b:array[0..rozmiarbufora] of integer;
    in,out:integer;
    n:integer;
    s:semaphore;{binarny do wzajemnego wykluczania}
    niepusty_sem,niepelny_sem:semaphore;{binarny}
    niepusty_licznik,niepelny_licznik:integer;

Procedure wloz(v:integer);
begin
    wait(s);
    if n=rozmiar_bufora+1 then
      begin
        niepusty_licznik:=niepusty_licznik+1;
        signal(s);
        wait(niepusty_sem);
        niepusty_licznik:=niepusty_licznik_1;
       end;
    ...wkładanie...
    if niepelny_licznik > 0 then signal (niepelny_sem)
    else signal(s)
end;
```

Symulacja Monitorów za pomocą Semaforów

```
Procedure pobierz(var v:integer);
begin
   wait(s);
      if n=0 then
      begin
         niepelny_licznik:=niepelny_licznik+1;
         signal(s);
         wait(niepelny_sem);
         niepelny_licznik:=niepelny_licznik-1;
      end;

      ...pobieranie....

if niepusty_licznik > 0 then signal(niepusty_sem)
else signal(s)
end;
```

Symulacja Monitorów za pomocą Semaforów

```
Procedure producent;
   produkuj, włóż itd...

Procedure konsument;
   pobierz; konsumuj; itd...

BEGIN
   in:=0;out:=0;n:=0;
   s:=1;
   niepusty_licznik:=0;niepelny_licznik:=0;
   niepusty_sem:=0;niepelny_sem:=0;
   cobegin;
      producent;konsument;
   coend;
END.
```

Sygnały swobodne – symulacja za pomocą semaforów

- Dotychczas było założenie że ostatnią instrukcją w monitorze był signal(zezwalający na wejście do monitora innym.
- teraz signal może być dowolnie wykonywany
- potrzeba wstrzymać proces w monitorze

Sygnały swobodne – symulacja za pomocą semaforów

- W modelu igloo wyglądało by to tak:
- oprócz zamrażarki stoi szafa z automatycznymi drzwiami
- proces który wykonał *signal* wszedł by do tej szafy
- w szafie jest stos czyli ostatni który wszedł pierwszy wypadnie
- gdy proces opuści pomieszczenie to drzwi szafy automatycznie się otwierają i wypada ostatni proces.
- ogólnie dbamy o to by w igloo nie przebywał więcej niż 1 proces.
- W szczególności sygnały swobodne mogą się sprowadzić do sygnału na końcu monitora, gdy po wyjściu z szafy od razu wyjdzie z igloo.
- nowa zmienna pilne i nowy semafor psem do liczenia i zawieszania procesów sygnalizujących.

Sygnały swobodne – symulacja za pomocą semaforów

wejście:

```
wait(s);
wait(war): liczwar:=liczwar:=liczwar+1;
           if pilne > 0 {jeżeli po signal czekają procesy }
              then signal(psem) {zwolnij taki proces}
           else signal(s); {wpuść nowy proces}
           wait(semwar);
           liczwar:=liczwar-1;
signal(war): pilne:=pilne+1;
           if liczwar > 0 then {jeżeli ktoś czeka}
           begin
              signal(semwar); {to wpuść go}
              wait(psem); {i zawieś swoje działanie }
           end;
           pilne:=pilne-1;
```

wyjście:

```
if pilne > 0 {jeżeli po signal czeka proces }
   then signal (psem) {zwolnij taki proces}
else signal(s); {wpuść nowy proces}
```

Problem 5 filozofów

- Ma wielkie znaczenie w badaniach nad programowaniem współbieżnym
- Może być wyzwaniem dla wszystkich tych którzy tworzą narzędzia pierwotne

Problem 5 filozofów

Założenia 5 filozofów

- żyło sobie 5 filozofów a filozofowie jak to na nich przystało myślą, a jak już się namyślą to jedzą.
- jedzenie mają przy jednym stole gdzie jest 5 nakryć, a pomiędzy nimi 5 pałeczek
- ale do jedzenia potrzebne są dwie pałeczki bo to filozofowie z dalekiego wschodu.

Problem 5 filozofów

```
Program pieciu_filozofow;
var paleczka:array[0..4] of semaphore; {b}
i:integer;

Procedure filozof (i:integer);
begin
   repeat
      mysl;
      wait(paleczka[i]);
      wait(paleczka[(i+1) mod 5]);
      jedz;
      signal(paleczka[i]);
      signal(paleczka[i+1) mod 5]);
   until false;
end;
BEGIN
for i:=0 to 4 do palkeczka[i] := 1;
cobegin
   filozof(0);filozof(1);...;filozof(4);
coend;
END.
```

To rozwiązanie dopuszcza blokadę. Przy nieszczęśliwym zbiegu okoliczności gdy wszyscy filozofowie wezmą pałeczki z jednej strony i będą czekać na drugie

Problem 5 filozofów

```
program pieciu_filozofow_v2;

monitor monitor_paleczka;
var paleczka:array[0..4] of integer;
   moznajesc:array[0..4] of condition;
i:integer;

Procedure wezpaleczke (i:integer);
begin
   if paleczka[i] <> 2 then wait (moznajesc[i] );
   paleczka[(i+1) mod 5] := paleczka[(i+1) mod 5] - 1;
   paleczka[(i-1) mod 5] := paleczka[(i-1) mod 5] - 1;

end;
```

Problem 5 filozofów

```
Procedure odloz_paleczke;
begin
   paleczka[(i+1) mod 5]:=paleczka[(i+1) mod 5] +1;
   paleczka[(i-1) mod 5]:=paleczka[(i-1) mod 5] +1;
   if paleczka[(i+1) mod 5 ] = 2 then signal (moznajesc[(i+1)
      mod 5]);
   if paleczka[(i-1) mod 5 ] = 2 then signal (moznajesc[(i-1)
      mod 5]);
end;

begin {monitor}
   for i:= to 4 do paleczka[i]:=2;
end;
```

Problem 5 filozofów

```
Procedure filozof (i:integer);
begin
   repeat
      mysl;
      wezpaleczke(i);
      jedz;
      odlozpaleczke(i);
   until false
end;


BEGIN
   cobegin
      filozof(0);filozof(1);…;filozof(4);
   coend;
END.
```

To rozwiązanie dopuszcza zagłodzenie, gdy na przemian to jeden to drugi proces dochodzą do zasobów a 3 tylko czeka.

Problem 5 filozofów

```
Program pieciu_filozofow;
var paleczka:array[0..4] of semaphore; {b}
pokoj: semaphore;
i:integer;

Procedure filozof(i:integer);
begin
    repeat
        mysl;
        wait(pokoj);
        wait(paleczka[i]);
        wait(paleczka[(i+1) mod 5]);
        jedz;
        signal(paleczka[i]);
        signal(paleczka[(i+1) mod 5]);
        signal(pokoj);
    until false
end;

BEGIN
    pokoj:=4;
    for i:=0 to 4 do paleczka[i]:=1;
    cobegin
        filozof(0);filozof(1);...;filozof(4);
    coend;
END.
```

Wprowadzamy dodatkowy semafor pokój, który gwarantuje że przy stole w jednej chwili może się znajdować co najwyżej 4 filozofów.
Z prostej zasady szufladkowej wynika że w jakiejkolwiek konfiguracji przynajmniej jeden filozof dostanie obie pałeczki.