Semafory

Rozwiązanie problemu wzajemnego wykluczania

- Bez dodatkowego wsparcia sprzętowego i programowego
  Zakładamy jedynie, że zapis do i odczyt z pamięci wspólnej są operacjami atomowymi (czyli istnieje arbiter wspólnej pamięci). W razie jednoczesnego zapisu i odczytu rezultatem będzie przeplecenie tych dwóch instrukcji w dowolnej kolejności
  Algorytm Dekkera → ćwiczenia!
  Wada:
  - złożoność
  - aktywne oczekiwanie
- Z dodatkowym wsparciem sprzętowym
  Specjalna instrukcja maszynowa realizująca atomowy zapis i odczyt, np.

```
Test_and_Set (Li)  begin
    Li := G;
    G := 1;
 end
```

```
var G: integer := 0;        // G - globalna dla wszystkich procesów

process P (i:integer);
   var Li: integer;         // Li - lokalna dla procesu i
loop
      sekcja_lokalna(i);
      loop                  // Wada - aktywne oczekiwanie
         Test_and_Set(Li);
          exit when Li = 0;
      end loop
      sekcja_krytyczna(i);
      G := 0;
end loop
```

- Wsparcie programowe: semafory, monitory itp.

Definicja (klasyczna) semafora

- Semafor jest zmienną całkowitoliczbową przyjmującą wartości nieujemne, dla której są określone tylko następujące operacje: inicjacja, wait (lub P), signal (lub V)
- Inicjacja, czyli nadanie wartości początkowej, jest wykonywane tylko raz dla danego semafora i poza procesami
- Operacje P i V są niepodzielne i wzajemnie wykluczające się (dla danego semafora)
- P(S): gdy S > 0, to S := S -1,
  wpp proces jest zawieszany w oczekiwaniu na ten warunek
  V(S): S := S + 1
- Oczekiwanie jest „sprawiedliwe”

Definicja (praktyczna) semafora

- P(S): jeśli S > 0, to S := S - 1, wpp wstrzymaj działanie procesu wykonującego tę operację
- V(S): jeśli są procesy wstrzymane w wyniku P(S), to wznów jeden z nich, wpp S := S + 1

Semafor binarny

- Jest szczególnym przypadkiem semafora ogólnego - przyjmuje jedynie wartości 1 i 0 (lub true i false)
- PB i VB są również niepodzielne i wzajemnie wykluczające się
- W definicji semafora ogólnego
  S := S + 1 przechodzi na S := 1;
  S := S - 1 przechodzi na S := 0
  (Uwaga: w niektórych implementacjach wykonanie VB(S) dla S = 1 kończy się błędem)
- Rozwiązanie problemu sekcji krytycznej (inicjalnie S = 1):

```
PB(S); sekcja_krytyczna(i); VB(S)
```

Różne uwagi

- P - Passeren, Proberen; V - Vrijmaken, Verhohen (Dijkstra jest Holendrem)
- Nie wolno testować wartości semafora (P, V i inicjacja to JEDYNE dopuszczalne operacje!)
- Operacje semaforowe wykluczają się wzajemnie, tak jak zapis i odczyt z tej samej komórki pamięci. Jednoczesne żądania wykonania operacji na tym samym semaforze będą wykonane sekwencyjnie, w nieznanej kolejności. W definicji semafora nie mówi się, który z zawieszonych procesów zostanie wznowiony; implementacja może (ale nie musi) wznawiać procesy zgodnie z zasadą kolejki prostej (żywotność ≠ sprawiedliwość)
- Implementacja semafora: zwykle na poziomie jądra systemu operacyjnego, bez aktywnego oczekiwania

Symulacja semaforów ogólnych binarnymi

(wg książki Shawa, WNT 1980, str. 90)
„Każdy semafor ogólny S może być zastąpiony przez zmienną całkowitą N_S i dwa semafory binarne $mutex_S$ i $delay_S$. Dla każdej operacji P(S) stosuje się wówczas:

```
P(mutexS); NS := Ns - 1;
if NS ≤ -1 then begin V(mutexS); P(delayS) end
else V(mutexS)
```

Każda zaś operacja V(S) jest zastępowana przez:

```
P(mutexS); NS := NS + 1;
if NS ≤ 0 then V(delayS);
V(mutexS)
```

Początkowo $mutex_S = 1$, $delay_S = 0$, a zmienna N_S przyjmuje początkową wartość zmiennej semaforowej S. ...”
Czy ta symulacja jest zawsze poprawna?

Zakładamy klasyczną definicję semafora

	P1	P2	P3	P4	*Poprawka Shawa*	
P(m_S);	1	5	9	13	P(m_S);	
$N_S := N_S - 1$;	2	6	10	14	$N_S := N_S - 1$;	
if $N_S \leq -1$ then	3	7	11	15	if $N_S \leq -1$ then	
begin					begin	
V(m_S);			12	16	V(m_S);	
P(d_S)					P(d_S)	
end					end;	$\Leftarrow$
else V(m_S);	4	8			V(m_S);	$\Leftarrow$
Sekcja krytyczna;						
P(m_S);	17	21			P(m_S);	
$N_S := N_S + 1$;	18	22			$N_S := N_S + 1$;	
if $N_S \leq 0$ then V(d_S);	19	23			if $N_S \leq 0$ then V(d_S)	
V(m_S)	20				else V(m_S)	$\Leftarrow$

N_S = 2 2:1 6:0 10:-1 14:-2 18:-1 22: 0 (operacja:wartość)

m_S = 1 1:0 4:1 5:0 8:1 9:0 12:1 13:0 16:1 17:0 20:1 21:0

d_S = 0 19:1 23:1 !!!

Przykład: Producent i konsument z ograniczonym buforem

```
S: semaphore := 1;
pełne, puste: semaphore := (0, N);    {liczniki miejsc w buforze}
process producent;
  loop
    produkuj;
    P(puste);   {sprawdź, czy bufor ma miejsca}
    P(S); wstaw; V(S);
    V(pełne);   {zwiększenie licznika pełnych miejsc}
  end loop
```

```
process konsument;
  loop
    P(pełne);  {sprawdź, czy bufor nie jest pusty}
    P(S); pobierz; V(S);
    V(puste);  {zwiększenie licznika pustych miejsc}
    konsumuj;
  end loop
```

Inne rodzaje semaforów

- Semafor dwustronnie ograniczony (Lipton 74)

  $P_k(S)$: if $S > 0$ then $S := S - 1$ else czekaj

  $V_k(S)$: if $S < k$ then $S := S + 1$ else czekaj

- Semafor OR (Lipton 74)

  $P_{OR}(S_1, ..., S_n)$: if $S_1 > 0 \vee ... \vee S_n > 0$ then $S_k := S_k - 1$ else czekaj
  gdzie $k = \min\{i: S_i > 0\}$

  $V_{OR}(S_k)$: $S_k := S_k + 1$

- Semafor AND (Patril 71)

  $P_{AND}(S_1, ..., S_n)$: if $S_1 > 0 \wedge ... \wedge S_n > 0$ then $\forall_k\ S_k := S_k - 1$
  else czekaj

  $V_{AND}(S_k)$: $S_k := S_k + 1$

- Semafor uogólniony (Lipton 74)

  $P(S_1, t_1, ..., S_n, t_n)$: if $S_1 \geq t_1 \wedge ... \wedge S_n \geq t_n$ then $\forall_k\ S_k := S_k - t_k$
  else czekaj

  $V(S_1, t_1, ..., S_n, t_n)$: $\forall_k\ S_k := S_k + t_k$

- Semafor sparametryzowany (Presser 75)

  $P(S_1, t_1, d_1, ..., S_n, t_n, d_n)$: if $S_1 \geq t_1 \wedge ... \wedge S_n \geq t_n$ then
  $\forall_k\ S_k := S_k - d_k$ else czekaj

  $V(S_1, d_1, ..., S_n, d_n)$: $\forall_k\ S_k := S_k + d_k$

Formalizm Habermanna

- Możliwość kontynuacji pracy przez dowolny proces przy wykonaniu operacji P zależy od liczby wykonanych w przeszłości (na tym semaforze) operacji P i V oraz od wartości początkowej semafora
- Oznaczenia:

  C(S) - wartość początkowa semafora S, $C(S) \geq 0$

  nV(S) - liczba wykonań operacji V na S

  nP'(S) - liczba wywołań operacji P na S (czyli ile razy procesy rozpoczęły operację P)

  nP(S) - liczba przejść przez operację P na S (czyli ile razy procesy mogły kontynuować pracę po wykonaniu operacji P)
- Wówczas

  P(S): nP'(S) := nP'(S) + 1
  <u>if</u> nP'(S) ≤ C(S) + nV(S) <u>then</u> nP(S) := nP(S) + 1

  V(S): <u>if</u> nP'(S) > C(S) + nV(S) <u>then</u> nP(S) := nP(S) + 1
  nV(S) := nV(S) + 1
- *Twierdzenie*

  $$nP(S) = \min(nP'(S), C(S) + nV(S))$$

  jest niezmiennikiem wykonań operacji P i V (dowód indukcyjny)

Semafory Agerwali

(T. Agerwala, „*Some extended semaphore primitives*", Acta Informatica, 1977, 8, 3)

- <u>Idea</u>: wprowadzenie jednoczesnych operacji semaforowych, w których wykonanie operacji typu P jest uzależnione od stanu podniesienia lub opuszczenia wskazanych semaforów

- PE(S_1, ..., S_n; S'_{n+1}, ..., S'_{n+m}) powoduje zawieszenie procesu do chwili, gdy dla wszystkich S_k ($k = 1,...,n$) będzie spełnione $S_k > 0$ oraz dla wszystkich S'_i ($i = n+1,...,n+m$) będzie spełnione $S'_i = 0$; <u>for</u> i := 1 <u>to</u> n <u>do</u> $S_i := S_i - 1$
- VE(S_1, ..., S_r): <u>for</u> i := 1 <u>to</u> r <u>do</u> $S_i := S_i + 1$
- S_k ($k = 1,...,n$); S'_i ($i=n+1,...,n+m$); S_t ($t = 1,...,r$) są semaforami
- Semafory Agerwali pozwalają na indywidualne traktowanie współpracujących ze sobą procesów, a więc na związanie z nimi priorytetów, podział na rozłączne klasy ze względu na prior. itp.
- Priorytetowy dostęp do zasobów: należy zsynchronizować dostęp N procesów do zasobu R. Dostęp ma być realizowany według priorytetów procesów (niższa wartość - wyższy priorytet)

```
var S: array[1..N] of semaphore := (0,...,0);
    T: semaphore := 1;
    R: resource;

process P (prio: 1..N)
  var k: 0..N;
begin
  k := prio - 1;
  loop
    praca bez zasobu;
    VE(S[prio]);
    if k = 0 then PE(T) else PE(T; S[1], ... , S[k]);
    PE(S[prio]);
    sekcja krytyczna z użyciem R;
    VE(T)
  end loop
end;

cobegin P(1); ... ; P(N) coend.
```

Problem pięciu filozofów - rozwiązanie z semaforami

- Rozwiązanie 1

w: array[0..4] of semaphore := (1,1,1,1,1)   {widelce}

```
process F (i:0..4)
begin
  loop
    myśli;
    P(w[i]); P(w[(i+1) mod 5]);
    je;
    V(w[i]); V(w[(i+1) mod 5]);
  end loop
end;
```

- Możliwość zastoju!

- Rozwiązanie 2

w: array[0..4] of semaphore := (1,1,1,1,1)   {widelce}
j: semaphore := 4;   {jadalnia}

```
process F (i:0..4)
begin
  loop
    myśli;
    P(j);
    P(w[i]); P(w[(i+1) mod 5]);
    je;
    V(w[i]); V(w[(i+1) mod 5]);
    V(j)
  end loop
end;
```

- Czy kolejność operacji P jest dowolna?
- Czy kolejność operacji V jest dowolna?
- „Super rejon krytyczny”

- Rozwiązanie 3

```
         0  i-ty filozof myśli
s[i]=   1  i-ty filozof jest głodny
         2  i-ty filozof je

s: array[0..4] of 0..2 := (0,0,0,0,0)   {stan}
m: semaphore := 1;     {mutex}
ps: array[0..4] of semaphore := (0,0,0,0,0)   {prywatny sem}

procedure test (k: 0..4);
begin
  if s[k] = 1 and s[(k-1) mod 5] ≠ 2 and s[(k+1) mod 5] ≠ 2
  then begin
    s[k] := 2;
    V(ps[k])
  end
end; {test}
process F (i:0..4)
begin
  loop
    myśli;
    P(m);
    s[i] := 1;
    test(i);
    V(m);
    P(ps[i]);
    je;
    P(m);
    s[i] := 0;
    test((i-1) mod 5]);
    test((i+1) mod 5]);
    V(m)
  end loop
end;
```

- Możliwość zagłodzenia

Semafory w systemie Unix

- Operacje na pojedynczym semaforze
  P(S,n) - opuszczenie S o wartość n
  V(S,n) - podniesienie S o wartość n
  Z(S) - czekaj, aż S = 0
  nP(S,n) - nieblokujące opuszczenie S o wartość n
  nV(S,n) - nieblokujące podniesienie S o wartość n

- Operacje jednoczesne
  [V(S1,1), P(S2,3), Z(S3)] - czekaj, aż $S2 \geq 3$ oraz $S3 = 0$ i wtedy S1 zwiększ o 1, a S2 zmniejsz o 3
  [nP(S1,1), Z(S2), V(S3,2)] - jeśli $S1 \geq 1$ i $S2 = 0$, to zmniejsz S1 o 1 i zwiększ S3 o 2, wpp nic nie rób (nieblokująca)
  Operacje nieblokujące dają w wyniku wartość logiczną **true**, gdy operacja się powiodła i **false** wpp

- Funkcje na semaforach
  wart(S) - wartość semafora S
  czekP(S) - liczba procesów czekających na P(S)
  czekZ(S) - liczba procesów czekających na Z(S)

  Uwaga: w języku C zapisuje się to inaczej