Podstawy programowania współbieżnego

Omawiane zagadnienia

1) Abstrakcja współbieżności
2) Problemy synchronizacyjne i narzędzia synchronizacji
3) Programowanie współbieżne w systemie Linux.
4) Komunikacja międzyprocesowa
5) Wprowadzenie do obliczeń równoległych

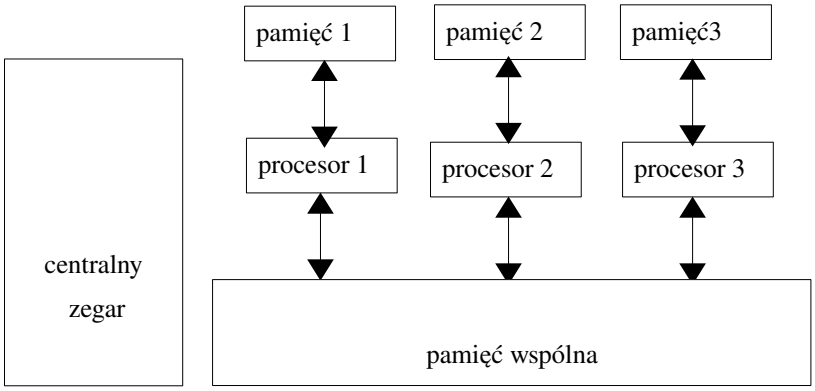
Literatura:

1. M. Ben-Ari, *Podstawy programowania współbieżnego i rozproszonego,* WNT, Warszawa 1996
2. Zbigniew Czech, *Wprowadzenie do obliczeń równoległych*

I

ABSTRAKCJA WSPÓŁBIEŻNOŚCI

1. Model komputera współbieżnego



Model komputera wieloprocesorowego (PRAM), na którym będą wykonywane programy komputerowe.

Cechy modelu PRAM :

- procesory pracują synchronicznie tzn. taktowane są wspólnym zegarem
- komunikacja między procesorami odbywa się poprzez pamięć wspólną
- wszystkie procesory mają stały czas dostępu do pamięci wspólnej
- niedopuszczalny jest wspólny zapis do tej samej komórki pamięci wspólnej przez kilka procesorów

Uwaga: w rzeczywistych komputerach wieloprocesorowych mogą występować odstępstwa od modelu PRAM. Może nie być spełnione założenie co do synchroniczności pracy procesorów. Również zapewnienie stałego czasy dostępu do wspólnej pamięci jest trudne do realizacji w przypadku dużej liczby procesorów.

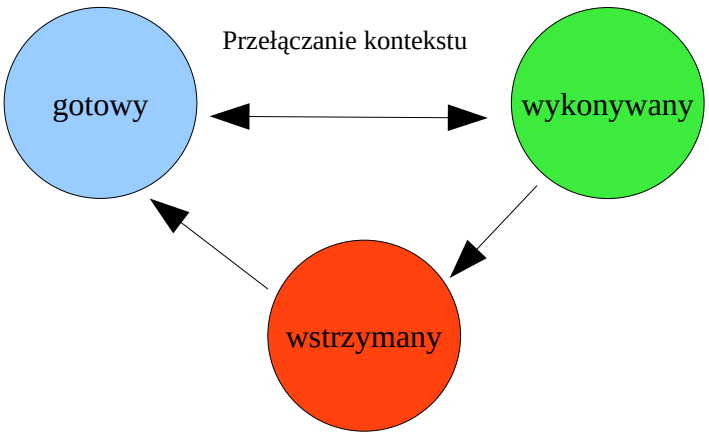
2. Podstawowe definicje i pojęcia

- proces – program załadowany do pamięci komputera

  Proces powstaje, gdy program zapisany na dysku załadujemy do pamięci komputera. Proces wykonuje instrukcje w ściśle określonej kolejności (sekwencji) tak jak zostały zapisane w programie. Proces nazywany jest również wątkiem lub zadaniem.

  stany procesu:

  a) wykonywany - gdy proces posiada procesor
  b) gotowy - gdy proces czeka na procesor
  c) wstrzymany - gdy proces otrzyma sygnał wstrzymania od sprzętu lub systemu (np. czeka na dane z wejścia)
  d) zakończony - stan w którym proces kończy działanie po otrzymaniu sygnału zakończenia lub wykonaniu operacji exit



Przejścia pomiędzy stanami procesu pokazuje diagram

Uwaga: Przejścia pomiędzy stanami gotowy i wykonywany realizuje system operacyjny przy pomocy mechanizmu, który się nazywa przełączaniem kontekstu. Przełączanie kontekstu pozwala na działanie wielu procesów w sytuacji, gdy ilość procesorów jest zbyt mała. W przypadku gdy liczba procesorów jest nie mniejsza niż liczba procesów wtedy przełączanie kontekstu nie występuje. Mówimy wtedy że procesy wykonują się równolegle.

- program sekwencyjny – program, w którym kolejność wykonywania instrukcji wynika z ich sekwencji zapisanej w programie. Program jest wykonywany przez jeden proces.

- program współbieżny – program składający się ze skończonej liczby programów sekwencyjnych, gdzie każdy program sekwencyjny wykonuje inny proces. W programie współbieżnym kolejność wykonywania instrukcji programu nie jest określona.

- przeplot - ciąg instrukcji programu współbieżnego wykonanych przez różne procesy i ułożonych według ich czasu wykonania. Wykonanie programu współbieżnego jest realizacją pewnego przeplotu zwanego scenariuszem.

- instrukcja atomowa – instrukcja, której wykonywanie nie może zostać przerwane

Scenariusze programów współbieżnych

program 1

$n \leftarrow 0$	
p1: $t \leftarrow n$	q1: $t \leftarrow n$
p2: $n \leftarrow t + 1$	q2: $n \leftarrow t+1$
p3: exit	q3: exit

Przedstawiony powyżej program współbieżny jest wykonywany przez dwa procesy *p* i *q*. Zmienna *n* jest w pamięci wspólnej (globalna), a zmienne t są lokalne dla procesów.

Określenie: Stan programu współbieżnego jest to element (krotka) zawierający po jednej etykiecie procesu i wartości zmiennych lokalnych i globalnych. Przykładowo dla powyższego programu mamy stan (p2:t=0,n=1; q3: t=0,n=1). Ten stan oznacza, że proces *p* jest gotowy do wykonania instrukcji *p2* (instrukcja p2 jest aktywna), a proces *q* jest gotowy do wykonania instrukcji *q3* (instrukcja q3 jest aktywna).

przykład

fragment diagramu stanów (obliczenie): (p1:t=?,n=0; q1:t=?,n=0) → (p2:t=0,n=0; q1:t=?,n=0) → (p3:t=0,n=1; q1:t=?,n=1) → (p3:t=0,n=1; q2:t=1,n=1) → (p3:t=0,n=2; q3:t=1,n=2)

	n=0	n=1	n=1	n=2	n=2	n=2
	t=0	t=0	t=0	t=0	t=0	t=0
	t=?	t=?	t=1	t=1	t=1	t=1
scenariusz:	p1-----	p2-------	q1------	q2------	p3------	q3

Wynik wykonania programu

scenariusz 1:	p1-p2-q1-q2-p3-q3	n= 2
scenariusz 2:	p1-q1-p2-q2-p3-q3	n= 1

Uwaga: wystąpienie w scenariuszu etykiety instrukcji oznacza, że ta instrukcja się zakończyła (została wykonana).

Obserwacja: różne scenariusze programu współbieżnego mogą dawać różne wyniki końcowe działania programu.

Wniosek: Wynikiem wykonania programu współbieżnego jest dowolny scenariusz.

Zauważyć: ciąg p2-q1-q2-p1-p3-q3 nie jest scenariuszem ponieważ proces *p* wykonuje instrukcje niezgodnie z kolejnością zapisu w programie (p2-p1-p3) .

Ćwiczenie1:

$n \leftarrow 0$

p1: powtarzaj 10 razy	q1:powtarzaj 10 razy
p2: $t \leftarrow n$	q2: $t \leftarrow n$
p3: $n \leftarrow t+1$	q3: $n \leftarrow t+1$
p4: exit	q4: exit

Uwaga: Tylko instrukcje wcięte (p2,p3 lub q2,q3) są podporządkowane pętli.
Zapis: $(p1\text{-}p2\text{-}p3)_2$ jest równoważny zapisowi (p1-p2-p3-p1-p2-p3)

Zapisać scenariusz który daje wartość zmiennej *n* równą:
a) 10
b) 2
Przykładowo scenariusz: $(p1\text{-}p2\text{-}p3)_{10}$-$(q1\text{-}q2\text{-}q3)_{10}$-p4-q4 daje n=20

Atomowość instrukcji

Instrukcje zapisane w języku asemblera lub maszynowym są atomowe. Program zapisany w językach wysokiego poziomu (np. język C) składa się z instrukcji, które na ogół nie są atomowe. Przykładowo instrukcja *n=n+1* lub *n++* z języka C rozpada się po kompilacji na instrukcje asemblerowe:

load R1,n
add R1,#1
store R1,n

W trakcie wykonywania tych instrukcji proces może nastąpić przełączenie kontekstu, np. po instrukcji *load* inny proces zacznie wykonywać swoje instrukcje co w konsekwencji może spowodować różny wynik instrukcji *n=n+1*.

Jako instrukcje atomowe można traktować jedynie instrukcje zapisu i odczytu z pamięci np. x=1, 1==x wykonywane na zmiennych pojedynczej precyzji. W przypadku innych typów zależy to od architektury procesora.

Zmienne krytyczne
Jeżeli dana zmienna jest przetwarzana w różnych procesach to nazywamy ją zmienną krytyczną. W instrukcji języka wysokiego poziomu zmienne krytyczne mogą występować po lewej stronie (przypisanie) lub po prawej stronie wyrażenia.
Wystąpienie krytyczne I rodzaju: zmienna jest modyfikowana w naszym procesie (występuje po lewej stronie instrukcji przypisania), a równocześnie występuje w innym procesie.
Wystąpienie krytyczne II rodzaju: zmienna występuje po prawej stronie w wyrażeniu w naszym procesie, a równocześnie jest modyfikowana w innym procesie.

Jeżeli wszystkie instrukcje programu współbieżnego zawierają co najwyżej pojedyncze wystąpienia krytyczne (I rodzaju lub II rodzaju) to taki program zachowuje się tak jak by te instrukcje były atomowe. Wynik działania takiego programu można przewidzieć badając przeploty utworzone z tych instrukcji.

Zauważ że wszystkie instrukcje w programie 1 mają co najwyżej pojedyncze wystąpienia krytyczne.

program 2

n ← 0	
p1: n ← n+1	q1: n ← n+1
p2: exit	q2: exit

W programie 2 instrukcje *p1* i *q1* nie są atomowe ponieważ zmienna *n* występuje podwójnie krytycznie w obu procesach.

program 3

n ← 0	
p1: n ← n+1	q1: n ← 2
p2: exit	q2: exit

W programie 3 instrukcja *p1* nie jest atomowa. Zmienna *n* występuje podwójnie krytycznie w instrukcji *p1*

program 4

n ← 0	
p1: t ← n	q1: n ← 2
p2: n ←t+1	q2: exit
p3: exit	

Program jest modyfikacją programu 3 polegającą na zastąpieniu instrukcji *p1* (w programie 3) dwoma instrukcjami *p1* i *p2*, które nie zawierają podwójnych wystąpień krytycznych.

program 5

n ← 0	
p1: n ← n+1	q1: x ← n
p2: exit	q2: exit

W programie 4 instrukcja *p1* może być traktowana jako atomowa, ponieważ jest tylko pojedyncze wystąpienie krytyczne zmiennej *n* w instrukcji *p1*.

3. Poprawność programów współbieżnych

Program sekwencyjny jest poprawny jeżeli spełnia własność *stopu* (kończy działanie) i daje poprawny wynik końcowy. Program współbieżny kończy działanie wtedy, gdy zakończą działanie wszystkie procesy.
Podczas wykonywania się programu współbieżnego są realizowane różne scenariusze, które mogą prowadzić do różnych wyników końcowych. Poprawność programu współbieżnego nie definiuje się więc za pomocą wyniku obliczeń, ale poprzez zachowanie pewnych własności podczas obliczeń.

własność bezpieczeństwa – wyrażenie logiczne (warunek logiczny), które jest prawdziwe w każdym stanie programu współbieżnego.

Naruszenie własności bezpieczeństwa prowadzi do nieprawidłowego działania programu współbieżnego.
Zakleszczenie (nieprawidłowe zatrzymanie procesów programu współbieżnego) jest również wynikiem naruszenia własności bezpieczeństwa.

własność żywotności – wyrażenie logiczne (warunek logiczny), które w końcu stanie się prawdziwe dla każdego scenariusza

Naruszenie własności żywotności prowadzi do wykluczenia procesów z dostępu do zasobów zwanego potocznie zagłodzeniem.

def. Program współbieżny jest poprawny, jeżeli spełnia obie własności: bezpieczeństwa i własność żywotności.

Słaba uczciwość - instrukcja aktywna(którą proces chce wykonać) zawsze się wykona

Uwaga: jeżeli instrukcja jest aktywna i nie wykona się do końca to taki scenariusz jest nie uczciwy (nie spełnia własności uczciwości słabej)

Przykład:

flaga ←false, n ←0

p1: while (flaga = false)	q1: n ←1
p2: n ←n+1	q2: flaga ←true
p3: exit	q3: exit

W scenariuszu: p1-q1-p2-$(p1\text{-}p2)_\infty$ proces *p* nigdy się nie zakończy. Ten scenariusz nie spełnia własności uczciwości słabej, ponieważ po wykonaniu instrukcji *q1* jest aktywna instrukcja *q2*, która w końcu musi być wykonana, w związku z czym powinna się pojawić w przeplocie. Takie scenariusze zawsze będą odrzucane.

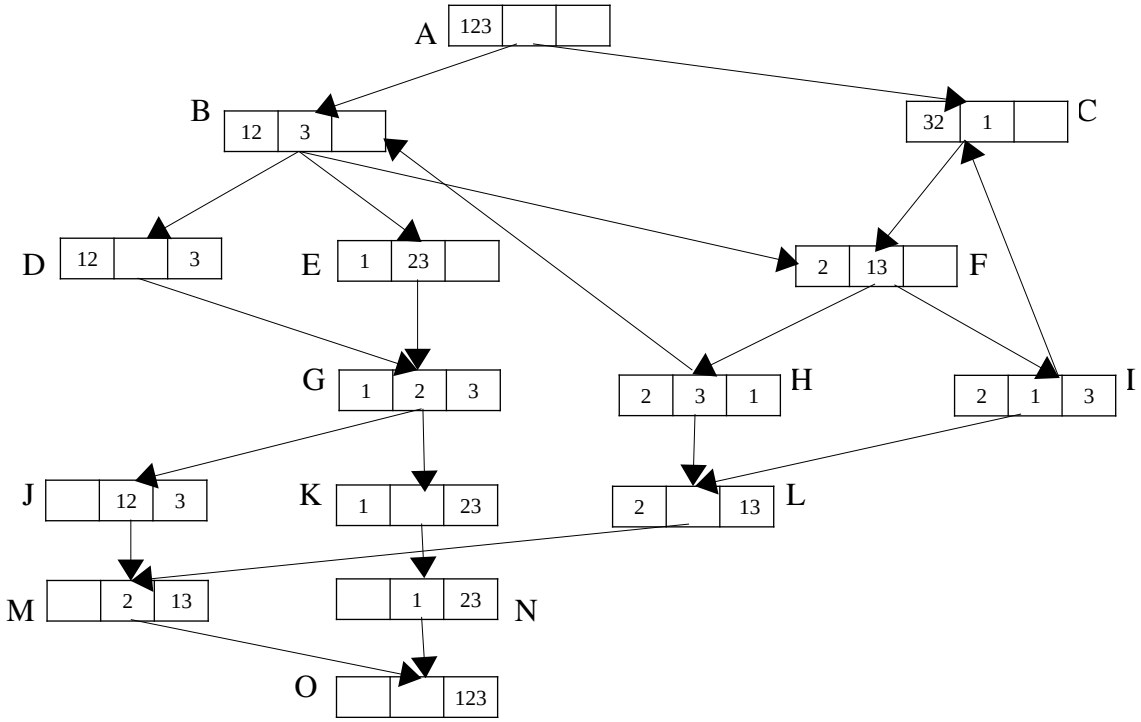
Def. Wynikiem wykonania programu współbieżnego jest dowolny uczciwy scenariusz.

Ćwiczenie2: Program współbieżny symuluje przejazd samochodów przez skrzyżowanie. Każdy samochód jest reprezentowany przez oddzielny proces. Samochody mogą poruszać się przez skrzyżowanie w kierunku północ – południe, albo wschód – zachód. Samochód po przejechaniu skrzyżowania może do niego wrócić inną drogą.

a) skonstruować diagram stanów dla tego programu

Diagram stanów jest przedstawiany za pomocą grafu skierowanego, którego węzły reprezentują stany programu współbieżnego. Diagram jest konstruowany w ten sposób, że wyznaczamy stan początkowy programu współbieżnego, a następnie dołączamy kolejne stany, w które przechodzi program wykonując pojedyncze akcje.

b) sformułować własność bezpieczeństwa i żywotności dla tego programu współbieżnego.



Fragment diagramu stanów dla programu symulującego przejazd przez skrzyżowanie. Element diagramu reprezentuje stan programu. Poszczególne pola oznaczają położenie samochodu odpowiednio dojazd do skrzyżowania, przejazd przez skrzyżowanie, opuszczenie skrzyżowania. Liczby w polach oznaczają procesy reprezentujące samochody znajdujące się w danym miejscu. Liczby nieparzyste oznaczają samochody jadące na kierunku wschód-zachód, a parzyste na kierunku północ-południe.

Definicje:

$s_1,s_2,\ldots,s_i,\ldots$ - ciąg stanów programu współbieżnego zwany obliczeniem

A – formuła (wyrażenie logiczne) *A* jest prawdziwa w stanie s_i

~A – formuła *A* jest nieprawdziwa w stanie s_i

1) $\Box A$ - formuła jest prawdziwa w stanie s_i danego obliczenia wtedy i tylko wtedy, gdy formuła A jest prawdziwa w wszystkich stanach s_j dla $j \geq i$ tego obliczenia

2) $\Diamond A$ - formuła jest prawdziwa w stanie s_i danego obliczenia wtedy i tylko wtedy, gdy formuła A jest prawdziwa w stanie s_j dla $j \geq i$

Własność bezpieczeństwa: $\Box$ ~((|12|) $\vee$ (|23|))

Własność żywotności: $\Diamond$ (||123)

Ćwiczenie3: Dane są trzy tablice zawierające liczby całkowite uporządkowane rosnąco. Tablice zawierają wspólną liczbę. Jakie będzie działanie procesu zapisanego poniżej w pseudokodzie:

```
                A[0..N], B[0..M], C[0..L]
                i ← j ← k ← 0
p1: powtarzaj
p2:     if A[i] < B[j]
p3:             i ←i+1
p4      else if B[j] < C[k]
p5:             j ←j+1
p6:     else if C[k] < A[i]
p7:             k ←k+1
p8:     else exit
```

Zapisać w pseudokodzie wersję współbieżną podanego programu.