Semafory

Semafory wykorzystujemy do synchronizacji wykonywania procesów i ich sekcji krytycznych. Z założenia w systemie wielozadaniowym wszystkie procesy mogą wykonywać się współbieżnie. Czasami jednak chcemy, by pewne procesy wykonały się w ustalonej z góry kolejności. Semafor to specjalnie traktowana przez system zmienna, która może przechowywać wartości całkowitoliczbowe {... –3, -2, -1, 0, 1, 2, 3 ...}. Semaforowi można tylko raz nadać wartość przez bezpośrednie przypisanie – przy inicjacji tej zmiennej. Potem dostęp do semafora jest możliwy tylko za pomocą dwóch niepodzielnych operacji: czekaj(SEMAFOR) i sygnalizuj(SEMAFOR).

Kod procesu P1:
jakiś kod
fragment kodu **S1**
jakiś kod

Kod procesu P2:
jakiś kod
fragment kodu **S2**
jakiś kod

Załóżmy, że mamy dane dwa procesy P1 i P2 oraz warunek, że proces P2 posiada fragment kodu S2 który musi wykonać się po fragmencie kodu S1 procesu P1. Zakładamy że oba procesy rozpoczynają pracę równolegle, np.: P1 na procesorze 1 a P2 na procesorze 2. Oba procesy pracują niezależnie aż do momentu, gdy P2 zechce wykonać fragment S2. S2 nie może zostać wykonany dopóki P1 nie wykona fragmentu S1. Musimy więc wstrzymać wykonywanie procesu P2 do momentu, aż P1 zakończy fragment S1. Natomiast P1, gdy wykona już cały fragment S1, powinien jakoś powiadomić P2, że ten może kontynuować swoją pracę i rozpocząć wykonywanie fragmentu S2. Do wymiany informacji między procesami posłuży właśnie SEMAFOR.

Deklarujemy zmienną SEMAFOR i przypisujemy jej wartość początkową 0. Następnie uruchamiamy współbieżnie oba procesy P1 i P2. Ustalmy, że dopóki SEMAFOR<=0, dopóty proces P2 nie będzie mógł wykonać fragmentu S2 - gdy P2 zechce wykonać S2 to będzie musiał poczekać na pozwolenie, czyli do momentu aż SEMAFOR>0. To pozwolenie wyda proces P1 zaraz po wykonaniu S1, zwiększając wartość SEMAFOR o jeden (po tej operacji SEMAFOR==1). Gdy tylko oczekujący na pozwolenie proces P2 zauważy, że SEMAFOR>0, wtedy zmniejszy wartość SEMAFOR o jeden i rozpocznie wykonywanie fragmentu S2.

Oczekiwanie w P2 na to, aż SEMAFOR będzie większy od zera, realizowane jest przez funkcję czekaj(SEMAFOR). Tak więc P2, by się upewnić przed przystąpieniem do wykonania S2 czy otrzymał już pozwolenie od P1, wywołuje funkcję czekaj(SEMAFOR). Funkcja ta odbiera kontrolę procesowi P2 (zawiesza go) do czasu zakończenia jej działania. W tym celu funkcja wykonuje pustą, nic nie robiącą pętlę dopóki SEMAFOR nie będzie mieć wartości większej od 0. Gdy warunek zakończenia pętli zostanie spełniony, funkcja czekaj(SEMAFOR) zmniejsza wartość SEMAFOR o jeden i kończy swoje działanie, zwracając kontrolę do procesu P2. W czasie, gdy P2 jest zawieszony P1 może spokojnie wykonać fragment S1. Po wykonaniu S1 proces P1 wywołuje funkcję sygnalizuj(SEMAFOR), która zwiększa wartość SEMAFOR o jeden.

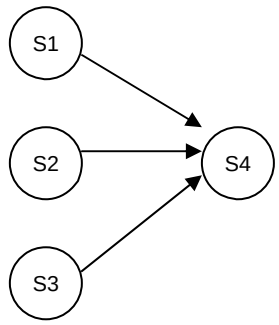
Deklaracja semafora:
SEMAFOR=0;

Kod procesu P1:
jakiś kod
fragment kodu **S1**
sygnalizuj(SEMAFOR);
jakiś kod

Kod procesu P2:
jakiś kod
czekaj(SEMAFOR);
fragment kodu **S2**
jakiś kod

Załóżmy, że mamy dane cztery procesy P1, P2, P3 i P4 zawierające odpowiednio instrukcje S1, S2, S3 i S4. Instrukcja S4 musi wykonać się po instrukcji S1, S2 i S3. Natomiast instrukcje S1, S2 i S3 nie powinny być synchronizowane.

Rysujemy graf skierowany:



Określamy relację w zbiorze wszystkich par instrukcji taką, że instrukcja Si jest w relacji z instrukcją Sj (Si, Sj) jeśli instrukcja Si poprzedza instrukcję Sj (to znaczy, że instrukcja Si musi wykonać się przed instrukcją Sj). Dla tak określonej relacji określamy macierz sąsiedztwa. Tyle matematyki. W macierzy w wierszu ‘i’ i w kolumnie ‘j’ wpiszemy 1, jeśli instrukcja Si musi wykonać się przed instrukcją Sj. W przeciwnym wypadku wpisujemy 0.

Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S2	S3	S4
S1	0	0	0	1
S2	0	0	0	1
S3	0	0	0	1
S4	0	0	0	0

Teraz trochę podobnie jak na układach logicznych. Próbujemy pokryć wszystkie jedynki (i tylko jedynki) jak najmniejszą liczbą prostokątów, które nie powinny na siebie zachodzić. Każdy prostokąt symbolizuje jeden semafor.

Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S2	S3	S4
S1	0	0	0	1
S2	0	0	0	1
S3	0	0	0	1
S4	0	0	0	0

Oznaczony na pomarańczowo obszar spełnia nasze wymogi. Pokryliśmy nim wszystkie jedynki. W związku z tym do zsynchronizowania procesów wystarczy nam tylko jeden semafor. Nazwijmy go SEMAFOR. Odpowiadający mu prostokąt rozciąga się na wiersze S1, S2 i S3 oraz na kolumnę S4. Oznacza to że instrukcje S1, S2 i S3 muszą zakończyć się zanim rozpocznie się S4. By to osiągnąć, każdy z procesów P1, P2 i P3 po zakończeniu, odpowiednio, instrukcji S1, S2 i S3 wywoła funkcję sygnalizuj(SEMAFOR). W procesie P4 przed instrukcją S4 wywołamy funkcję czekaj(SEMAFOR). Musimy teraz określić wartość początkową zmiennej SEMAFOR.

Deklaracja semafora:
SEMAFOR=???;

Kod procesu P1:	**Kod procesu P2:**	**Kod procesu P3:**	**Kod procesu P4:**
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*
fragment kodu **S1**	fragment kodu **S2**	fragment kodu **S3**	czekaj(SEMAFOR);
sygnalizuj(SEMAFOR);	sygnalizuj(SEMAFOR);	sygnalizuj(SEMAFOR);	fragment kodu **S4**
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*

Nadajemy zmiennej SEMAFOR pewną szukaną przez nas wartość początkową. Uruchamiamy równolegle wszystkie cztery procesy. Załóżmy, że proces P4 doszedł do instrukcji S4 i chce ją wykonać, a żadna z instrukcji S1, S2 i S3 nie została jeszcze wykonana. Proces P4 wywołuje funkcję czekaj(SEMAFOR) by sprawdzić, czy ma pozwolenie i w razie porażki zawiesić swoje działanie do czasu, aż takie pozwolenie nadejdzie. Pozwolenia nie ma, więc proces P4 rozpoczyna czekanie na funkcji czekaj(SEMAFOR). Ale aby funkcja czekaj(SEMAFOR) mogła zawiesić proces P4, musi być spełniony warunek SEMAFOR<=0. Czyli wartość początkowa SEMAFOR podana przy deklaracji tej zmiennej jest z pewnością mniejsza lub równa zero.

Powiedzmy że proces P1 wykonał swoją instrukcję S1 jako pierwszy. P1 po zakończeniu S1 wywołał funkcję sygnalizuj(SEMAFOR) (zwiększenie wartości SEMAFOR o jeden). Mimo to proces P4 powinien nadal czekać, ponieważ nie wykonały się jeszcze instrukcje S2 ani S3. Czyli nadal powinien być spełniony warunek: SEMAFOR<=0. Powiedzmy że następną instrukcją, jaka się wykonała, jest S2 procesu P2. Proces P2 wywołuje funkcję sygnalizuj(SEMAFOR), co powoduje zwiększenie wartości SEMAFOR o jeden. Nadal jednak proces P3 nie wykonał instrukcji S3, więc proces P4 powinien czekać, co oznacza że SEMAFOR<=0. Przyszła kolej na S3. Po wykonaniu instrukcji S3 proces P3 wywołuje, podobnie jak procesy P1 i P2, instrukcję sygnalizuj(SEMAFOR). SEMAFOR zostaje zwiększony o jeden. Wykonały się wszystkie instrukcje, na które czekał proces P4 przed wykonaniem instrukcji S4. Teraz proces P4 powinien zakończyć oczekiwanie i przejść do wykonywania instrukcji S4. Jak pamiętamy funkcja czekaj(SEMAFOR) odblokuje proces P4 dopiero wtedy, gdy SEMAFOR>0. Wtedy funkcja zakończy wykonywać pustą pętlę, zmniejszy SEMAFOR o jeden i zwróci sterowanie do procesu który ją wywołał, czyli do P4. By to było możliwe, po wykonaniu S1, S2 i S3 oraz znajdujących się za nimi funkcji sygnalizuj(SEMAFOR), wartość zmiennej SEMAFOR powinna wynosić jeden, czyli SEMAFOR==1. To oznacza, że początkowa wartość zmiennej SEMAFOR musiała być równa –2.

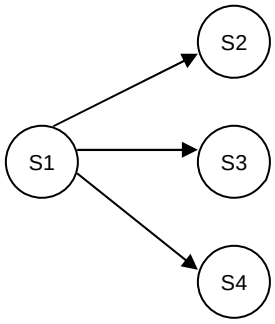
Deklaracja semafora:
SEMAFOR=-2;

Kod procesu P1:	Kod procesu P2:	Kod procesu P3:	Kod procesu P4:
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*
fragment kodu **S1**	fragment kodu **S2**	fragment kodu **S3**	czekaj(SEMAFOR);
sygnalizuj(SEMAFOR);	sygnalizuj(SEMAFOR);	sygnalizuj(SEMAFOR);	fragment kodu **S4**
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*

Sprawdźmy to, analizując poprzednio podaną sytuację. Na początku SEMAFOR=-2. Jako pierwszy do swojego fragmentu S4 doszedł proces P4. Wywołał funkcję czekaj(SEMAFOR). W związku z tym, że SEMAFOR==-2, funkcja czekaj(SEMAFOR) weszła w pustą pętlę i tym samym zawiesiła proces P4 do momentu gdy SEMAFOR>0. Po pewnym czasie proces P1 wykonał S1 i wywołał sygnalizuj(SEMAFOR). Wartość zmiennej SEMAFOR wzrosła z –2 do –1. Proces P4 nadal czeka, bo SEMAFOR<=0. Następnie proces P2 wykonał S2 i wywołał sygnalizuj(SEMAFOR). Wartość zmiennej SEMAFOR wzrosła z –1 do 0. Proces P4 nadal czeka, bo SEMAFOR<=0. Następnie proces P3 wykonał S3 i wywołał sygnalizuj(SEMAFOR). Wartość zmiennej SEMAFOR wzrosła z 0 do 1. Teraz funkcja czekaj(SEMAFOR) wywołana w procesie P4 może zakończyć działanie, bo SEMAFOR>0. Funkcja kończy wykonywanie pustej pętli, zmniejsza wartość zmiennej SEMAFOR o jeden, czyli z 1 do 0 i zwraca kontrolę do procesu P4. Proces P4 może nareszcie wykonać swój fragment S4.

Załóżmy, że mamy dane cztery procesy P1, P2, P3 i P4 zawierające odpowiednio instrukcje S1, S2, S3 i S4. Instrukcje S2, S3 i S4 muszą wykonać się po instrukcji S1. Natomiast instrukcje S2, S3 i S4 nie powinny być synchronizowane.

Rysujemy graf skierowany:



Wyznaczamy tabelkę relacji poprzedzający-poprzedzany i pokrywamy wszystkie jedynki (i tylko jedynki) jak najmniejszą liczbą prostokątów (które nie powinny na siebie zachodzić):

Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S2	S3	S4
S1	0	1	1	1
S2	0	0	0	0
S3	0	0	0	0
S4	0	0	0	0

Teraz zastosujemy szybką metodę określenia, które z instrukcji S1, S2, S3 i S4 powinny być zakończone funkcjami sygnalizuj() a które poprzedzone operacjami czekaj().

Instrukcja S1 poprzedza instrukcje S2, S3 i S4. Mamy jeden semafor; nazwijmy go SEMAFOR. Ponieważ S2, S3 i S4 czekają na S1, to przed każdą z nich będzie wywoływana instrukcja czekaj(SEMAFOR). Każde wywołanie funkcji czekaj(SEMAFOR) spowoduje po jej zakończeniu zmniejszenie wartości SEMAFOR o jeden. Przed zakończeniem S1 wartość SEMAFOR powinna być mniejsza równa zero, aby procesy P2, P3 i P4, gdy dojdą do swoich instrukcji S, weszły w stan oczekiwania. Gdy S1 się zakończy, powinniśmy tak zmodyfikować wartość SEMAFOR, aby każda z instrukcji S2, S3 i S4 miała możliwość się wykonać. Niech początkowo SEMAFOR=0. Będzie to spełniać powyższe założenia. Po wykonaniu S1 proces P1 powinien trzykrotnie wywołać funkcję sygnalizuj(SEMAFOR). Dzięki temu SEMAFOR zostanie trzykrotnie zwiększony o jeden, czyli z 0 do 3. Teraz SEMAFOR>0, czyli każdy z procesów może się obudzić. Powiedzmy, że jako pierwszy kontrolę odzyskał proces P2. Zanim jednak funkcja czekaj(SEMAFOR) odpowiedzialna za dotychczasowe uśpienie procesu P2 zakończyła działanie, zdążyła zmniejszyć wartość SEMAFOR o jeden, czyli z 3 do 2. Proces P2 może teraz wykonać fragment S2. Podobnie będzie dla kolejnego procesu (wartość SEMAFOR spadnie do 1) i dla ostatniego (wartość SEMAFOR spadnie do 0). Dla każdego z budzących się procesów spełniony był warunek, że SEMAFOR>0.

Deklaracja semafora:
SEMAFOR=0;

Kod procesu P1:	**Kod procesu P2:**	**Kod procesu P3:**	**Kod procesu P4:**
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*
fragment kodu **S1**	czekaj(SEMAFOR);	czekaj(SEMAFOR);	czekaj(SEMAFOR);
sygnalizuj(SEMAFOR); sygnalizuj(SEMAFOR); sygnalizuj(SEMAFOR);	fragment kodu **S2**	fragment kodu **S3**	fragment kodu **S4**
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*

Przyjrzyjmy się jeszcze raz tabelce relacji i tabelce kodów poszczególnych procesów. Jeżeli założymy, że po ostatecznym „zużyciu” semafor musi mieć wartość 0, to otrzymujemy prosty wzór na wartość początkową semafora:
SEMAFOR=0-(liczba wywołań funkcji sygnalizuj)+(liczba wywołań funkcji czekaj)
czyli:
SEMAFOR=(liczba wywołań funkcji czekaj)-(liczba wywołań funkcji sygnalizuj)

Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S2	S3	S4	
S1	0	1	1	1	po wykonaniu S1 trzykrotnie wywołaj sygnalizuj(SEMAFOR)
S2	0	0	0	0	
S3	0	0	0	0	
S4	0	0	0	0	
		Przed S2 wywołaj czekaj(SEMAFOR)	Przed S3 wywołaj czekaj(SEMAFOR)	Przed S4 wywołaj czekaj(SEMAFOR)	Początkowo SEMAFOR=(1+1+1)-3=0

Deklaracja semafora:
SEMAFOR=0;

Kod procesu P1:	**Kod procesu P2:**	**Kod procesu P3:**	**Kod procesu P4:**
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*
fragment kodu **S1**	czekaj(SEMAFOR);	czekaj(SEMAFOR);	czekaj(SEMAFOR);
sygnalizuj(SEMAFOR); sygnalizuj(SEMAFOR); sygnalizuj(SEMAFOR);	fragment kodu **S2**	fragment kodu **S3**	fragment kodu **S4**
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*

A to przypomnienie pierwszego przykładu oraz wyjaśnienie:

Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S2	S3	S4	
S1	0	0	0	1	po wykonaniu S1 jednokrotnie wywołaj sygnalizuj(SEMAFOR)
S2	0	0	0	1	po wykonaniu S2 jednokrotnie wywołaj sygnalizuj(SEMAFOR)
S3	0	0	0	1	po wykonaniu S3 jednokrotnie wywołaj sygnalizuj(SEMAFOR)
S4	0	0	0	0	
				Przed S4 wywołaj czekaj(SEMAFOR)	Początkowo SEMAFOR=1-(1+1+1)=-2

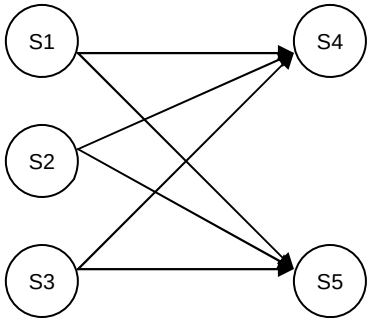
Deklaracja semafora:
SEMAFOR=-2;

Kod procesu P1:	Kod procesu P2:	Kod procesu P3:	Kod procesu P4:
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*
fragment kodu **S1**	fragment kodu **S2**	fragment kodu **S3**	czekaj(SEMAFOR);
sygnalizuj(SEMAFOR);	sygnalizuj(SEMAFOR);	sygnalizuj(SEMAFOR);	fragment kodu **S4**
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*

Wszystko można odczytać z powyższych tabelek. Załóżmy, że badamy warunki tylko dla prostokąta pomarańczowego, określającego SEMAFOR. Powiedzmy, że wybieramy wiersz dla S1, czyli sprawdzamy, jakie instrukcje są poprzedzane przez S1. Suma jedynek w obrębie wiersza S1 (przy założeniu że nie wychodzimy poza pomarańczowy prostokąt) wynosi 1. To znaczy że S1 poprzedza jedną instrukcję. Dlatego po S1 musi wystąpić jednokrotne wywołanie funkcji sygnalizuj(SEMAFOR). Podobnie dla wierszy S2 i S3. Teraz zabieramy się do analizowania kolumn. S4 czeka na chwilę, gdy SEMAFOR>0, czyli przed S4 musi wystąpić wywołanie funkcji czekaj(SEMAFOR). Wartość początkowa SEMAFOR=(liczba wywołań funkcji czekaj)-(liczba wywołań funkcji sygnalizuj)=1-(3*1)=-2. Teraz pozostaje udowodnić, dlaczego to jest poprawne zsynchronizowanie (patrz poprzednie opisy).

Rozważmy następujące zadanie:

Załóżmy, że mamy danych pięć procesów P1, P2, P3, P4 i P5 zawierających odpowiednio instrukcje S1, S2, S3, S4 i S5. Instrukcje S4 i S5 muszą wykonać się po instrukcjach S1, S2 i S3. Natomiast instrukcje S1, S2 i S3 nie powinny być synchronizowane. Także instrukcje S4 i S5 nie powinny być zsynchronizowane.



Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S2	S3	S4	S5	
S1	0	0	0	1	1	po wykonaniu S1 dwukrotnie wywołaj sygnalizuj(SEMAFOR)
S2	0	0	0	1	1	po wykonaniu S2 dwukrotnie wywołaj sygnalizuj(SEMAFOR)
S3	0	0	0	1	1	po wykonaniu S3 dwukrotnie wywołaj sygnalizuj(SEMAFOR)
S4	0	0	0	0	0	
S5	0	0	0	0	0	
				Przed S4 wywołaj czekaj(SEMAFOR)	Przed S5 wywołaj czekaj(SEMAFOR)	Początkowo SEMAFOR=2-6=-4

Deklaracja semafora:
SEMAFOR=-4;

Kod procesu P1:
jakiś kod
fragment kodu **S1**
sygnalizuj(SEMAFOR); sygnalizuj(SEMAFOR);
jakiś kod

Kod procesu P2:
jakiś kod
fragment kodu **S2**
sygnalizuj(SEMAFOR); sygnalizuj(SEMAFOR);
jakiś kod

Kod procesu P3:
jakiś kod
fragment kodu **S3**
sygnalizuj(SEMAFOR); sygnalizuj(SEMAFOR);
jakiś kod

Kod procesu P4:
jakiś kod
czekaj(SEMAFOR);
fragment kodu **S4**
jakiś kod

Kod procesu P5:
jakiś kod
czekaj(SEMAFOR);
fragment kodu **S5**
jakiś kod

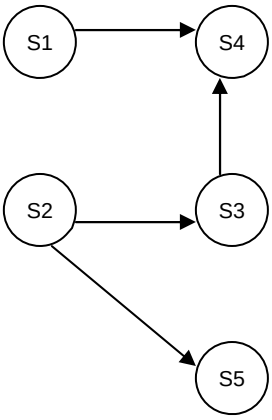
By zadanie było w pełni rozwiązane, powinniśmy opisać kolejne kroki działania procesów. Czyli:

Wartość początkowa SEMAFOR=-4. Procesy P1, P2, P3, P4 i P5 zaczynają wykonywać się współbieżnie. Załóżmy, że jako pierwszy do fragmentu S doszedł P4. SEMAFOR==-4 czyli SEMAFOR<=0 więc P4 wchodzi w stan oczekiwania (do momentu, aż SEMAFOR>0). Niech następnym procesem, który doszedł do fragmentu S będzie P5. SEMAFOR==-4 czyli SEMAFOR<=0 więc P5 wchodzi w stan oczekiwania (do momentu, aż SEMAFOR>0). Niech następnym procesem, który doszedł do fragmentu S będzie P1. P1 wykonuje fragment S1 i dwukrotnie wywołuje sygnalizuj(SEMAFOR), co powoduje zwiększenie SEMAFOR o dwa, z –4 do –2. Procesy P4 i P5 nadal czekają, bo SEMAFOR<=0. Niech następnym procesem,

który doszedł do fragmentu S będzie P2. P2 wykonuje fragment S2 i dwukrotnie wywołuje sygnalizuj(SEMAFOR), co powoduje zwiększenie SEMAFOR o dwa, z –2 do 0. Procesy P4 i P5 nadal czekają, bo SEMAFOR<=0. Niech następnym procesem, który doszedł do swojego fragmentu S będzie P3. P3 wykonuje fragment S3 i dwukrotnie wywołuje sygnalizuj(SEMAFOR), co powoduje zwiększenie SEMAFOR o dwa, z 0 do 2. Procesy P4 i P5 mogą wreszcie wykonać swoje fragmenty S, bo SEMAFOR>0. Powiedzmy, że jako pierwszy rozważamy proces P4. Proces P4 kończy czekać na funkcji czekaj(SEMAFOR), bo SEMAFOR==2. Funkcja czekaj(SEMAFOR) spowodowała zmniejszenie wartości SEMAFOR o jeden, z 2 do 1. Teraz P4 wykonuje fragment S4. Zajmijmy się teraz procesem P5. Proces P5 kończy czekać na funkcji czekaj(SEMAFOR), bo SEMAFOR==1. Funkcja czekaj(SEMAFOR) spowodowała zmniejszenie wartości SEMAFOR o jeden, z 1 do 0. Teraz P5 wykonuje fragment S5. Wszystkie instrukcje S zostały wykonane w odpowiedniej kolejności, czyli zastosowana synchronizacja jest w porządku. Trzeba pamiętać, że podana powyżej kolejność dochodzenia procesów do swoich fragmentów S jest przykładowa. Jeżeli synchronizacja jest poprawna, to dla każdej kombinacji instrukcje S wykonają się we właściwej kolejności.

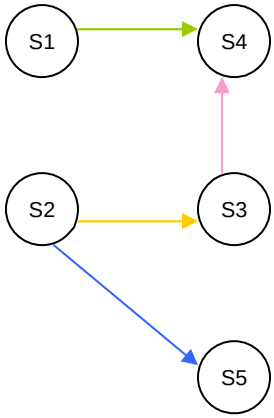
Rozważmy następujące zadanie:

Załóżmy, że mamy danych pięć procesów P1, P2, P3, P4 i P5 zawierających odpowiednio instrukcje S1, S2, S3, S4 i S5. Instrukcja S3 musi wykonać się po instrukcji S2. Instrukcja S4 musi wykonać się po instrukcji S1 i S3. Instrukcja S5 musi wykonać się po instrukcji S2.

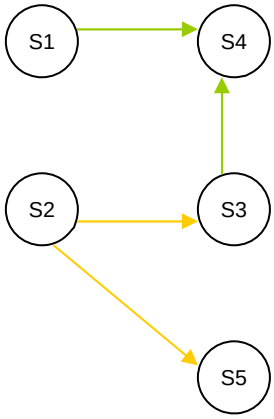


Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S2	S3	S4	S5
S1	0	0	0	1	0
S2	0	0	1	0	1
S3	0	0	0	1	0
S4	0	0	0	0	0
S5	0	0	0	0	0



Jeśli wnikliwiej przyjrzymy się grafowi, to zauważymy że wystarczą tylko dwa semafory:



Czy to znaczy, że według tabelki musimy użyć aż cztery semafory, chociaż z rysunku grafu wynika, że wystarczą tylko dwa? W odpowiedzi przychodzi nam teoria relacji z baz danych. Zamiana kolejności dowolnych dwóch kolumn lub dowolnych dwóch wierszy nie ma żadnego wpływu na relację. Możemy to robić bezkarnie.

Zamieńmy miejscami kolumnę S2 z kolumną S5:

Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S5	S3	S4	S2
S1	0	0	0	1	0
S2	0	1	1	0	0
S3	0	0	0	1	0
S4	0	0	0	0	0
S5	0	0	0	0	0

Następnie zamieńmy miejscami wiersz S4 z wierszem S1:

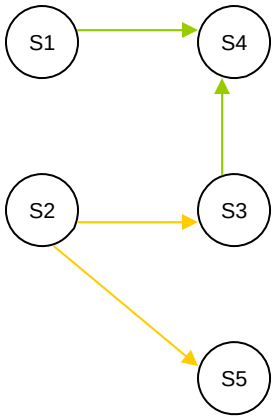
Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S5	S3	S4	S2
S4	0	0	0	0	0
S2	0	1	1	0	0
S3	0	0	0	1	0
S1	0	0	0	1	0
S5	0	0	0	0	0

Sprawdźmy, czy nic się nie zepsuło: S5 czeka na S2, S3 czeka na S2, S4 czeka na S1 i S3. Czyli wszystko zgodnie z treścią zadania. Dokonujemy nowego pokrycia jedynek prostokątami:

Instrukcje poprzedzane

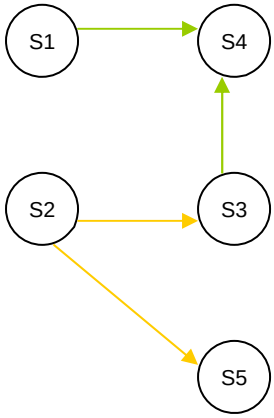
Instrukcje poprzedzające	S1	S5	S3	S4	S2
S4	0	0	0	0	0
S2	0	1	1	0	0
S3	0	0	0	1	0
S1	0	0	0	1	0
S5	0	0	0	0	0



Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S5	S3	S4	S2	
S4	0	0	0	0	0	
S2	0	1	1	0	0	Po wykonaniu S2 dwukrotnie sygnalizuj(SEM_POMARAŃCZ)
S3	0	0	0	1	0	Po wykonaniu S3 jednokrotnie sygnalizuj(SEM_ZIELONY)
S1	0	0	0	1	0	Po wykonaniu S1 jednokrotnie sygnalizuj(SEM_ZIELONY)
S5	0	0	0	0	0	
		Przed S5 wywołaj czekaj(SEM_POMARAŃCZ)	Przed S3 wywołaj czekaj(SEM_POMARAŃCZ)	Przed S4 wywołaj czekaj(SEM_ZIELONY)		

Wartości początkowe semaforów:
SEM_POMARAŃCZ=2-2=0
SEM_ZIELONY=1-2=-1



Deklaracja semaforów:
SEM_POMARAŃCZ=0;
SEM_ZIELONY=-1;

Kod procesu P1:
jakiś kod
fragment kodu **S1**
sygnalizuj(SEM_ZIELONY);
jakiś kod

Kod procesu P2:
jakiś kod
fragment kodu **S2**
sygnalizuj(SEM_POMARAŃCZ); sygnalizuj(SEM_POMARAŃCZ);
jakiś kod

Kod procesu P3:
jakiś kod
czekaj(SEM_POMARAŃCZ)
fragment kodu **S3**
sygnalizuj(SEM_ZIELONY);
jakiś kod

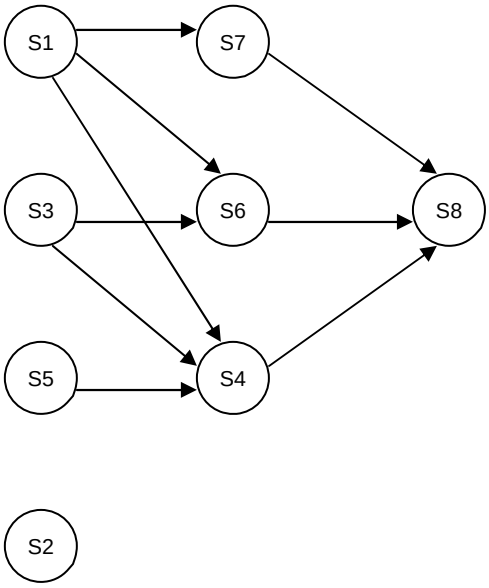
Kod procesu P4:
jakiś kod
czekaj(SEM_ZIELONY);
fragment kodu **S4**
jakiś kod

Kod procesu P5:
jakiś kod
czekaj(SEM_POMARAŃCZ);
fragment kodu **S5**
jakiś kod

Teraz piszemy, dlaczego to wszystko działa (patrz poprzednie przykłady)...

Rozważmy następujące zadanie (wersja XXL, której na pewno nie będzie na egzaminie, mimo to przeczytaj ją, bo ujmuje razem wszystkie dotychczas wprowadzone metody):

Załóżmy, że mamy danych osiem procesów P1, P2, P3, P4, P5, P6, P7 i P8 zawierających odpowiednio instrukcje S1, S2, S3, S4, S5, S6, S7 i S8. Instrukcja S7 musi wykonać się po instrukcji S1. Instrukcja S4 musi wykonać się po instrukcjach S1, S3 i S5. Instrukcja S6 musi wykonać się po instrukcjach S1 i S3. Instrukcja S8 musi wykonać się po instrukcjach S4, S6 i S7. Nie należy synchronizować instrukcji S1, S3 i S5. Nie należy synchronizować instrukcji S4, S6 i S7.



Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S2	S3	S4	S5	S6	S7	S8
S1	0	0	0	1	0	1	1	0
S2	0	0	0	0	0	0	0	0
S3	0	0	0	1	0	1	0	0
S4	0	0	0	0	0	0	0	1
S5	0	0	0	1	0	0	0	0
S6	0	0	0	0	0	0	0	1
S7	0	0	0	0	0	0	0	1
S8	0	0	0	0	0	0	0	0

Analizujemy tabelkę. Patrzymy jak można poprzestawiać kolumny i wiersze, aby otrzymać jak najmniejszą liczbę prostokątów pokrywających:

Instrukcje poprzedzane

Instrukcje poprzedzające

	S1	S2	S3	S4	S5	S6	S7	S8
S1	0	0	0	1	0	1	1	0
S2	0	0	0	0	0	0	0	0
S3	0	0	0	1	0	1	0	0
S4	0	0	0	0	0	0	0	1
S5	0	0	0	1	0	0	0	0
S6	0	0	0	0	0	0	0	1
S7	0	0	0	0	0	0	0	1
S8	0	0	0	0	0	0	0	0

Zamieniamy miejscami kolumny S5 i S6:

Instrukcje poprzedzane

Instrukcje poprzedzające

	S1	S2	S3	S4	S6	S5	S7	S8
S1	0	0	0	1	1	0	1	0
S2	0	0	0	0	0	0	0	0
S3	0	0	0	1	1	0	0	0
S4	0	0	0	0	0	0	0	1
S5	0	0	0	1	0	0	0	0
S6	0	0	0	0	0	0	0	1
S7	0	0	0	0	0	0	0	1
S8	0	0	0	0	0	0	0	0

Zamieniamy miejscami wiersze S2 i S3:

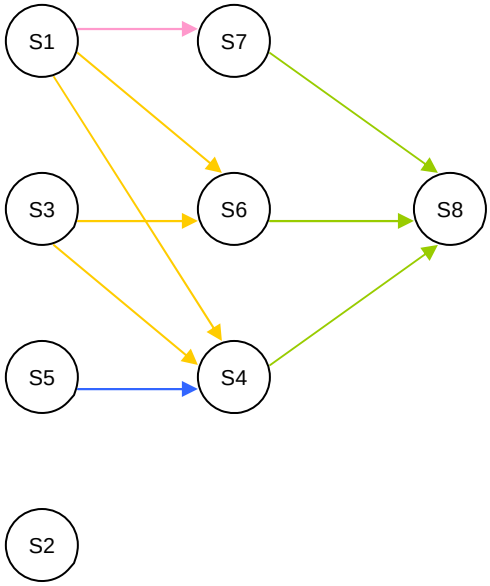
Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S2	S3	S4	S6	S5	S7	S8
S1	0	0	0	1	1	0	1	0
S3	0	0	0	1	1	0	0	0
S2	0	0	0	0	0	0	0	0
S4	0	0	0	0	0	0	0	1
S5	0	0	0	1	0	0	0	0
S6	0	0	0	0	0	0	0	1
S7	0	0	0	0	0	0	0	1
S8	0	0	0	0	0	0	0	0

Zamieniamy miejscami wiersze S4 i S5:

Instrukcje poprzedzane

Instrukcje poprzedzające	S1	S2	S3	S4	S6	S5	S7	S8
S1	0	0	0	1	1	0	1	0
S3	0	0	0	1	1	0	0	0
S2	0	0	0	0	0	0	0	0
S5	0	0	0	1	0	0	0	0
S4	0	0	0	0	0	0	0	1
S6	0	0	0	0	0	0	0	1
S7	0	0	0	0	0	0	0	1
S8	0	0	0	0	0	0	0	0



Instrukcje poprzedzane

Instrukcje poprzedzające

	S1	S2	S3	S4	S6	S5	S7	S8
S1	0	0	0	1	1	0	1	0
S3	0	0	0	1	1	0	0	0
S2	0	0	0	0	0	0	0	0
S5	0	0	0	1	0	0	0	0
S4	0	0	0	0	0	0	0	1
S6	0	0	0	0	0	0	0	1
S7	0	0	0	0	0	0	0	1
S8	0	0	0	0	0	0	0	0

Po zakończeniu S1 dwukrotnie sygnalizuj(SEM_POM) i jednokrotnie sygnalizuj(SEM_RÓŻ)

Po zakończeniu S3 dwukrotnie sygnalizuj(SEM_POM)

Po zakończeniu S5 jednokrotnie sygnalizuj(SEM_NIEB)

Po zakończeniu S4 jednokrotnie sygnalizuj(SEM_ZIEL)

Po zakończeniu S6 jednokrotnie sygnalizuj(SEM_ZIEL)

Po zakończeniu S7 jednokrotnie sygnalizuj(SEM_ZIEL)

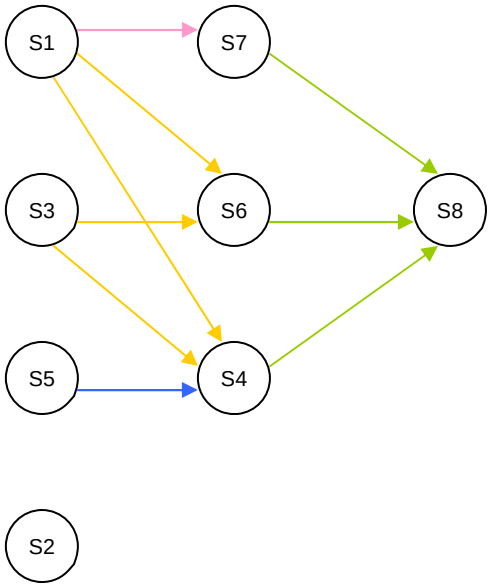
Przed S4 wywołaj czekaj(SEM_POM) i czekaj(SEM_NIEB)

Przed S6 wywołaj czekaj(SEM_POM)

Przed S7 wywołaj czekaj(SEM_RÓŻ)

Przed S8 wywołaj czekaj(SEM_ZIEL)

Wartości początkowe:
SEM_POM=(1+1)-(2+2)=-2
SEM_ZIEL=1-(1+1+1)=-2
SEM_NIEB=1-1=0
SEM_RÓŻ=1-1=0



Deklaracja semaforów:
SEM_POM=-2
SEM_ZIEL=-2
SEM_NIEB=0
SEM_RÓŻ=0

Kod procesu P1:	**Kod procesu P2:**	**Kod procesu P3:**	**Kod procesu P4:**
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*
			czekaj(SEM_POM); czekaj(SEM_NIEB);
fragment kodu **S1**	fragment kodu **S2**	fragment kodu **S3**	fragment kodu **S4**
sygnalizuj(SEM_POM); sygnalizuj(SEM_POM); sygnalizuj(SEM_RÓŻ);		sygnalizuj(SEM_POM); sygnalizuj(SEM_POM);	sygnalizuj(SEM_ZIEL);
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*

Kod procesu P5:	**Kod procesu P6:**	**Kod procesu P7:**	**Kod procesu P8:**
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*
	czekaj(SEM_POM);	czekaj(SEM_RÓŻ);	czekaj(SEM_ZIEL);
fragment kodu **S5**	fragment kodu **S6**	fragment kodu **S7**	fragment kodu **S8**
sygnalizuj(SEM_NIEB);	sygnalizuj(SEM_ZIEL);	sygnalizuj(SEM_ZIEL);	
jakiś kod	*jakiś kod*	*jakiś kod*	*jakiś kod*