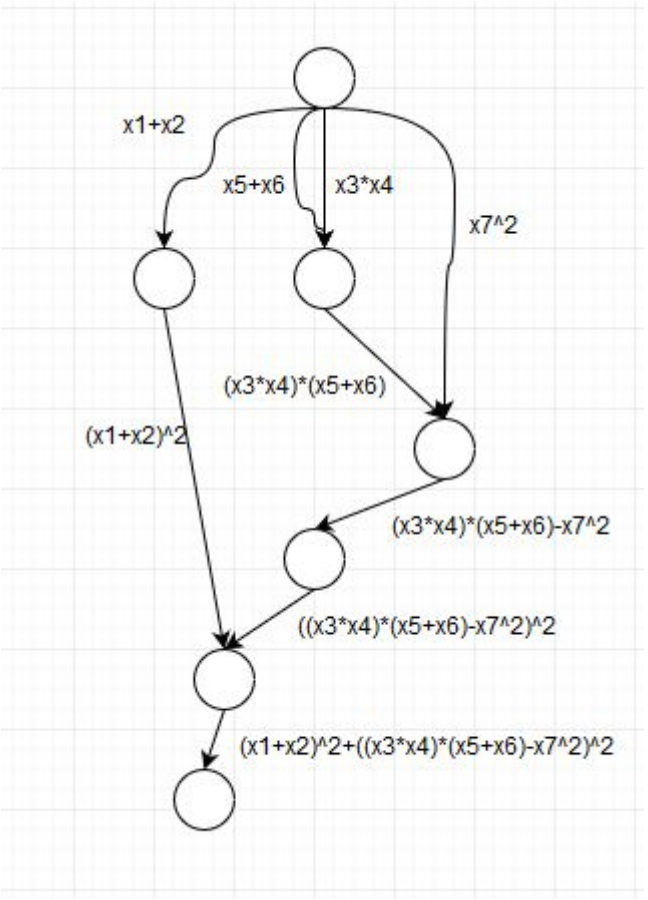
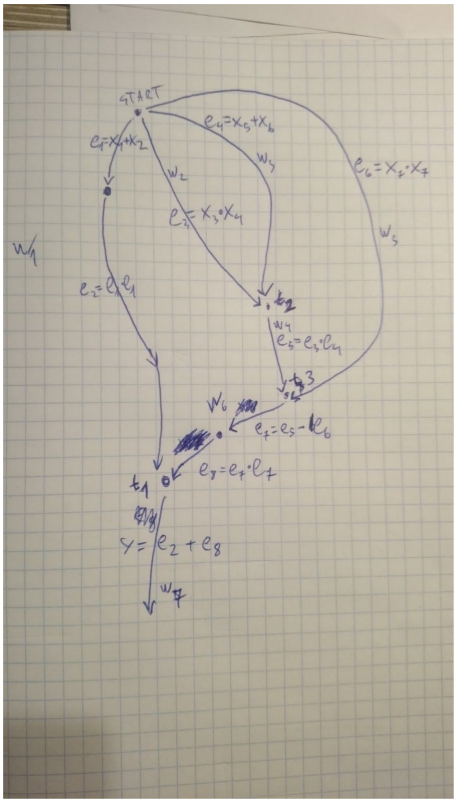
Przedstaw optymalny graf przepływu procesów oraz zgodne z nim programy współbieżne zaimplementowane z użyciem operacji „*and*” oraz „*fork-join-quit*”, odpowiadające współbieżnemu wyznaczaniu wartości następującego wyrażenia:

$$y = (x_1 + x_2)^2 + \left((x_3 * x_4)(x_5 + x_6) - x_7^2\right)^2$$

```
begin
    t1 := 2;
     t2 := 2;
     t3 := 2;
     fork w1;
     fork w2;
     fork w3;
     fork w5;
     quit;

w1:
    e1 = x1 + x2;
    e2 = e1 * e1;
    join t1, w7;
    quit;

w2 :
    e3 = x3 * x4;
    join t2, w4;
    quit;

w3:
    e4 = x5 + x6;
    joint t2, w4;
    quit;

w4:
    e5 = e3 * e4;
    join t3, w6;
    quit

w5:
    e6 = x7 * x7;
    join t3, w6;
    quit;

w6 :
    e7 = e5 - e6;
    e8 = e7 * e7;
    join t1, w7l
    quit;

w7:
    y = e2 + e8;
    quit;
end.
```






```
begin
   begin
   e1 = x1 + x2
   e2 = e1 * e1
   end
and
   begin
      begin
         e3 = x3 * x4
      and
         e4 = x5 + x6
      e5 = e3 * e4
      end
   and
      e6 = x7 * x7
   e7 = e5 - e6
   e8 = e7 * e7
   end
y = e2 + e8
end
```

Dany jest stan początkowy systemu:

$$A=\begin{bmatrix}2&1&3&2\\3&2&2&2\\1&2&1&1\end{bmatrix}\quad m=\begin{bmatrix}11\\12\\11\end{bmatrix}\quad C=\begin{bmatrix}6&7&6&4\\6&6&5&4\\5&6&8&5\end{bmatrix}$$

a) Stosując algorytm Holta sprawdź, czy stan jest bezpieczny.

Podobne zadanie rozwiazane

$$H=\begin{bmatrix}4&6&3&2\\3&4&3&2\\4&4&7&4\end{bmatrix}\quad E=\begin{bmatrix}2&3&4&6\\2&3&3&4\\4&4&4&7\end{bmatrix}$$

$$f_0=\begin{bmatrix}3\\3\\6\end{bmatrix}\quad l_0=\begin{bmatrix}1\\1\\1\end{bmatrix}\quad c_0=[4\ 3\ 3\ 3\ 3]$$

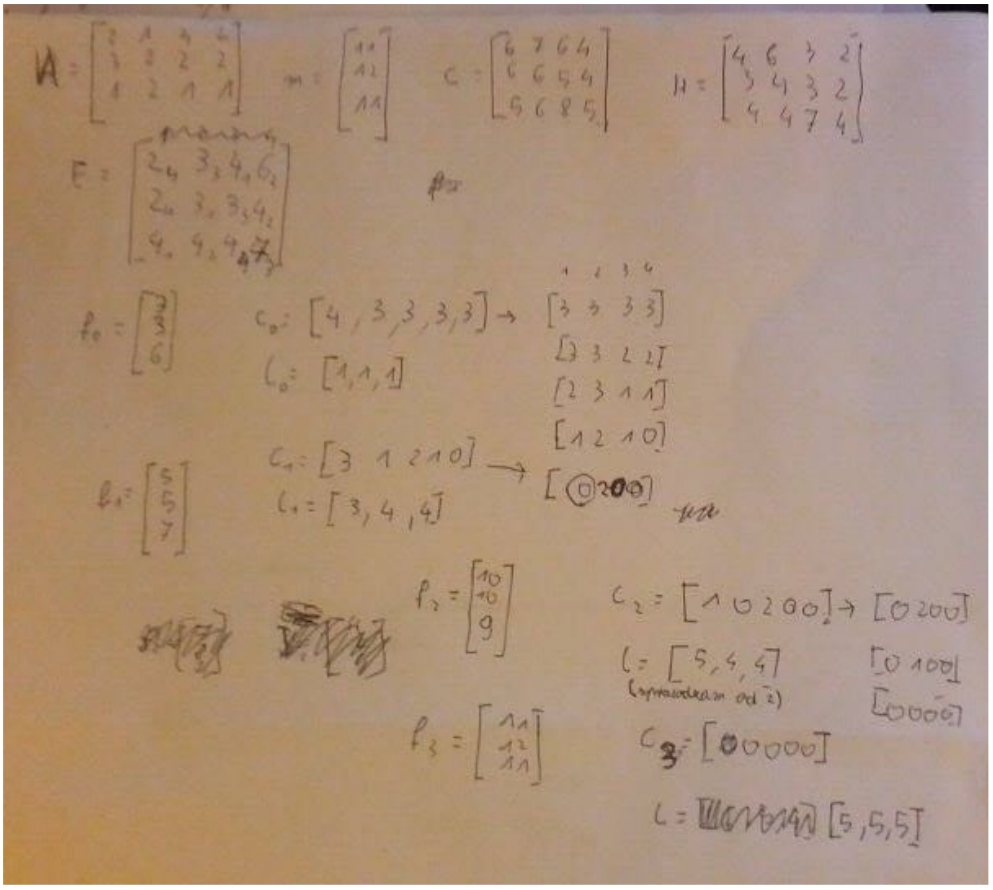
$$f_1=\begin{bmatrix}5\\5\\7\end{bmatrix}\quad l_1=\begin{bmatrix}2\\3\\3\end{bmatrix}\quad c_1=[3\ 1\ 2\ 1\ 0]$$

$$f_2=\begin{bmatrix}10\\10\\9\end{bmatrix}\quad l_2=\begin{bmatrix}3\\4\\4\end{bmatrix}\quad c_2=[1\ 0\ 1\ 0\ 0]$$

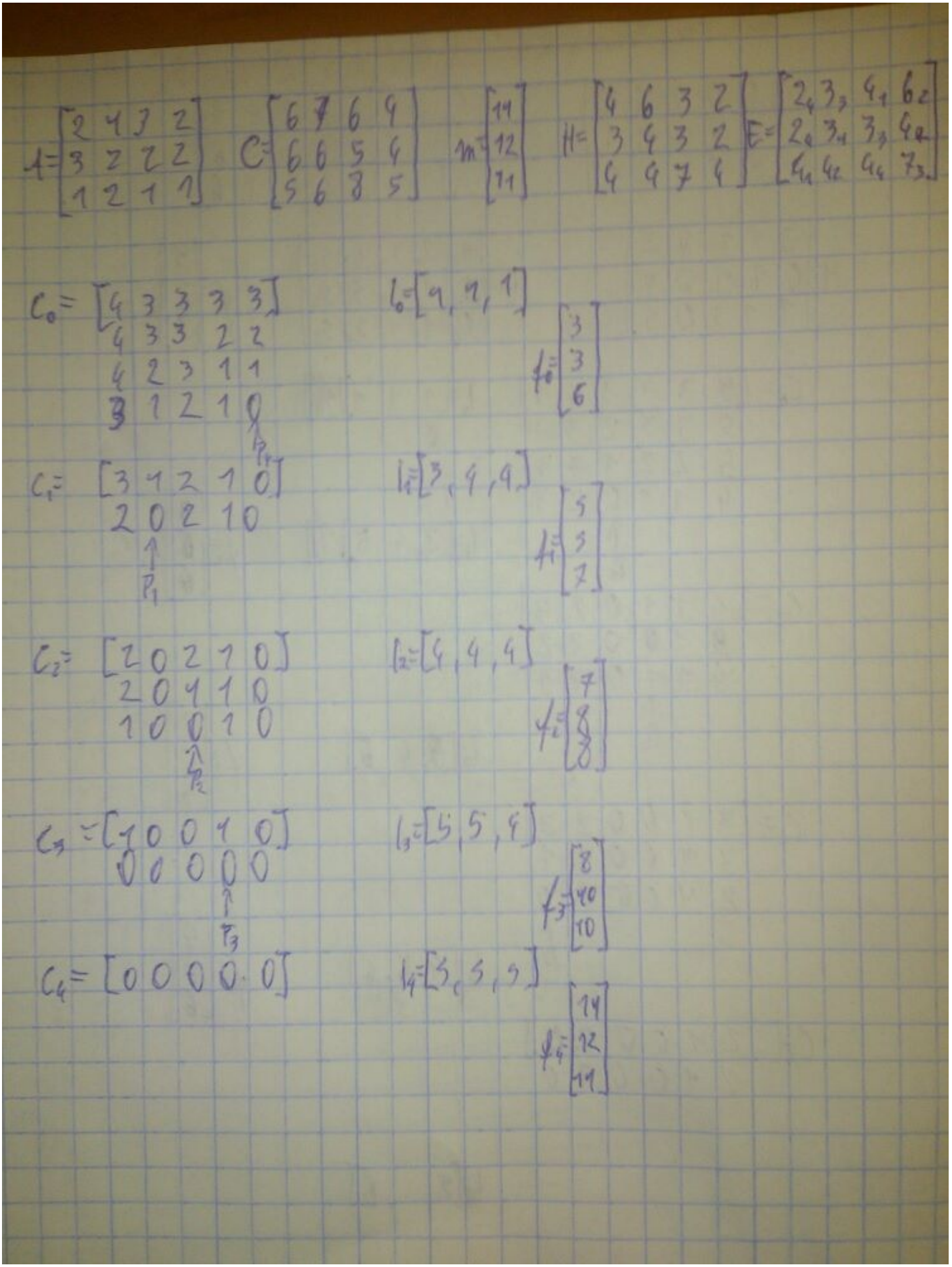
$$f_3=\begin{bmatrix}11\\12\\11\end{bmatrix}\quad l_3=\begin{bmatrix}4\\4\\4\end{bmatrix}\quad c_3=[0\ 0\ 0\ 0\ 0]\rightarrow \text{Stan bezpieczny}$$

D

coś takiego? f1 = [6,5,7] wg mnie , jest ok zwalniany jest ostatni zasób [2,2,1] + [3,3,6] = [5,5,7]
^to jest chyba źle c2 = [1 0 2 0 0]? nie powinno tak byc?
Wartości w l1-3 powinny być zwiększone o dupa
Też mam wrażenie że tam jest błąd, jak się popatrzy na przykład Brzezińskiego to przejście zawsze opuszcza licznik dla zadań o jeden, niezależnie czy mogę więcej zadań “zakończyć” w tej pętli.
To moja propozycja rozwiązania, może ktoś się wypowie - :JEST OK



Wstawiam moje rozwiązanie:



^ to jest źle chyba bo I nie powinno przekroczyć hm … wartości 5, bo inicjujemy 1 i max można dodać do 5, potem ten stan nie jest brany pod uwagę
Jeśli masz na myśli I4 to tam jest pośpieszna czcionka - [5,5,5] //a sorry mój błąd :)

wyjaśni ktoś dlaczego c2 = [1, 0, 0, 1, 0] a nie [1, 0, 1, 0, 0] ? //rozpatrujemy zasób nr 2 czyli w macierzy E wiersz 2 kolumna $l_2[2]=4$ jest wartość 4 dla procesu 2 którą zaspokajamy dostępnym zasobem $f_2[2]=8$ (ten zasób dla procesu nr 3 jest już zaspokojony)

b) Zakładając, że konsekwentnie stosowane jest podejście unikania, podaj sekwencję stanów systemu (zaznaczając procesy zawieszone) odpowiadającą następującego ciągowi żądań zasobów:

- w chwili $t_1 : p^a(P_1) = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$
- w chwili $t_2 > t_1 : p^a(P_2) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
- w chwili $t_3 > t_2 : p^r(P_4) = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$

prawie identyczne zadanie rozwiazane dałoby radę wytłumaczyć te chwile, w których momentach algo mamy to wykonać?chodzi chyba o to że t1 potem t2 nast t3.tak, ale w którym konkretnym momencie to wykonać? Wydaje mi się że trzeba założyć że pomiędzy naszymi alokacjami/zwolnieniami zasobów danymi w zadaniu w systemie nic się nie dzieje. No a zwolnienia zasobów które są realizowane podczas próby alokacji są czysto hipotetyczne.

I zakładamy przydzielenie zasobów dla P_1 i mamy

$$A=\begin{bmatrix}4&1&3&2\\4&2&2&2\\1&2&1&1\end{bmatrix} \quad H=\begin{bmatrix}2&6&3&2\\2&4&3&2\\4&4&7&4\end{bmatrix} \quad C=\begin{bmatrix}6&7&6&4\\6&6&5&4\\5&6&8&5\end{bmatrix} \quad m=\begin{bmatrix}11\\12\\11\end{bmatrix}$$

$$f_0=\begin{bmatrix}1\\2\\6\end{bmatrix} \qquad \rho^a(P_1)=\begin{bmatrix}2\\1\\0\end{bmatrix}$$

sytuacja niebezpieczna, nie jesteśmy w stanie zapewnić kompletu zasobów żadnemu z procesów – nie przydzielamy zasobów do P_1

II zakładamy przydzielenie zasobów do procesu P_2 i mamy

$$A=\begin{bmatrix}2&2&3&2\\3&3&2&2\\1&3&1&1\end{bmatrix} \quad H=\begin{bmatrix}4&5&3&2\\3&3&3&2\\4&3&7&4\end{bmatrix} \qquad \rho^a(P_2)=\begin{bmatrix}1\\1\\1\end{bmatrix}$$

$$f_0=\begin{bmatrix}2\\2\\9\end{bmatrix} \quad f_1=\begin{bmatrix}4\\4\\6\end{bmatrix} \quad f_2=\begin{bmatrix}6\\7\\7\end{bmatrix} \quad f_3=\begin{bmatrix}8\\10\\10\end{bmatrix} \quad f_4=\begin{bmatrix}11\\12\\11\end{bmatrix}$$

$P_2 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3$

~~Sytuacja bezpieczna~~ Przydzielenie jest bezpieczne (realizujemy)

III Proces P_2 może zwolnić zasoby w wyniku czego otrzymamy:

$$A=\begin{bmatrix}2&2&3&0\\3&3&2&1\\1&3&1&0\end{bmatrix} \qquad \rho^a(P_2)=\begin{bmatrix}2\\1\\1\end{bmatrix}$$

IV spróbujmy ponownie przydzielić zasoby na żądanie P_1

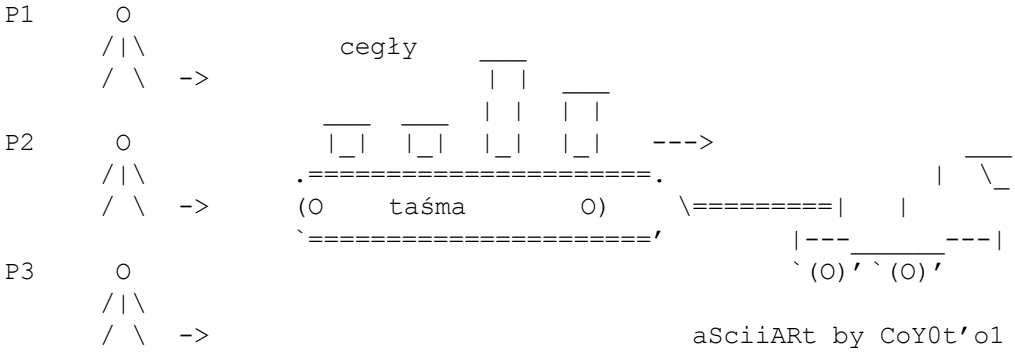
$$A=\begin{bmatrix}4&2&3&0\\4&3&2&1\\1&3&1&0\end{bmatrix} \quad H=\begin{bmatrix}2&5&3&4\\2&3&3&3\\4&3&7&5\end{bmatrix} \qquad \rho^a(P_1)=\begin{bmatrix}2\\1\\0\end{bmatrix}$$

$$f_0=\begin{bmatrix}2\\2\\6\end{bmatrix} \quad f_1=\begin{bmatrix}6\\6\\7\end{bmatrix} \quad f_2=\begin{bmatrix}8\\9\\10\end{bmatrix} \quad f_3=\begin{bmatrix}11\\11\\11\end{bmatrix} \quad f_4=\begin{bmatrix}11\\12\\11\end{bmatrix}$$

$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4$

więc możemy bezpiecznie przydzielić zasoby do P_1

Przy taśmie transportowej pracuje trzech pracowników oznaczonych przez P_1, P_2 i P_3. Pracownicy wrzucają na taśmę cegły o masach odpowiednio 1, 2 i 3 jednostki. Na końcu taśmy stoi ciężarówka o ładowności C jednostek, którą należy zawsze załadować do pełna. Wszyscy pracownicy starają się układać cegły na taśmie najszybciej jak to możliwe. Taśma może przetransportować w danej chwili maksymalnie K sztuk cegieł. Jednocześnie jednak taśma ma ograniczony udźwig: maksymalnie M jednostek masy, tak, że niedopuszczalne jest położenie np. samych tylko cegieł najcięższych ($3K>M$). Po zapełnieniu ciężarówki na jej miejsce pojawia się natychmiast nowa o takich samych parametrach. Cegły „zjeżdżające" taśmy *muszą* od razu trafić na samochód dokładnie w takiej kolejności jak zostały położone na taśmie.



Napisz programy symulujące działanie pracowników i ciężarówki. Do implementacji programów można wykorzystać uogólnione operacje semaforowe P i V operujące na semaforach wielowartościowych (atomowe zmniejszanie i zwiększanie semafora o dowolną wartość). Proszę użyć następującej notacji: $P(s,n)$ oznacza opuszczenie semafora s o n jednostek, przy czym $P(s)\equiv P(s,1)$.

Przykład

Załóżmy, że C=5, K=3 i M=4. Cegły mogłyby być położone na taśmie w następującej kolejności:

```
   3, 1, _, 1, 2, 1, 1, 1, _, 3, _, _, 2,  <-- trzecia ciężarówka
a
   |        |           |
  początek  |           druga ciężarówka
            |
            pierwsza ciężarówka zapełniona
```

Gdzie „_" oznacza puste miejsce na taśmie (nieobsadzone z powodu możliwego przeciążenia taśmy). Każde 3(K) sąsiednie pozycje nie powinny mieć masy w sumie większej od 4 (M).

```
1  var
2      TE : binary semaphore;
3      TC, TM : semaphore;
4      //X - waga cegły
5      //K - ilość cegieł na taśmie
6
7  procedure zaladuj_cegle(X: INTEGER)
8  begin
9      P(TC, X);           //TC - ciężarówka
10     P(TE);              //TE - miejsce przy taśmie
11     P(TM, X);           //TM - udźwig taśmy
12     poloz_cegle();
13     delay(1);           //czekanie, aż się przesunie
14     V(TE);              //zwolnienie miejsce przy taśmie
15     delay(K-1);         //cegła dojeżdża do końca taśmy
16     T += X;             //pakujemy cegłę
17     V(TM, X);           //cegła już nie jest na taśmie
18     if T == C then      //jeśli ciężarówka pełna
19     begin
20         wywiez_cegly();
21         V(TC, C);       //podstawiamy nową ciężarówkę
22     end;
23 end;
24
25 begin
26     TC = C;
27     TM = M;
28 end;
```

```
1  procedure P1;
2  begin
3      while true do
4      begin
5          produkuj_cegle(1);
6          zaladuj_cegle(1);
7      end;
8  end;
9
10 procedure P2;
11 begin
12     while true do
13     begin
14         produkuj_cegle(2);
15         zaladuj_cegle(2);
16     end;
17 end;
18
19 procedure P3;
20 begin
21     while true do
22     begin
23         produkuj_cegle(3);
24         zaladuj_cegle(3);
25     end;
26 end;
```

TE : semaphore;
w sekcji init: TE = 1;

Ma ktoś coś lepszego od tego (to nie jest idealne)?
//A czy to nie jest tak, że mają być tylko operacje P i V?
// Ja bym jeszcze dodał semafor binarny na sekcje T += X, aby kika procesów nie mogło robić tego jednocześnie

Przedstaw implementację wielowartościowych operacji semaforowych $P(S,n)$ i $V(S,n)$ z użyciem binarnych operacji semaforowych $P_b(B)$ i $V_b(B)$.

// przykładowy autorski kod, proszę o weryfikację

```
var
mutex, binarny, delay: BINARY SEMAPHORE;
N, oczekiwana: INTEGER;
oczekuje: BOOLEAN;

procedure P(value: INTEGER)
begin
	P(mutex);
	NS := N;
	N:= N - value;
	if N >= 0 then
	begin

		V(mutex);
	end
	else
	begin
		P(delay);
		oczekiwana := NS;
		oczekuje := true;
		V(mutex);
		while oczekiwana > N do P(binarny);
		V(delay);
	end;
end;

procedure V(value: INTEGER)
begin
	P(mutex);
	N := N + value;
	if oczekuje then
	begin
		oczekuje := false;
		V(binarny);
	end;
	V(mutex);
end;

begin
	mutex := 1; // zapewnia anonimowość
	binarny := 0; // główny semafor binarny

	delay := 1; // blokada oczekiwania dla P gdy wartość NS ujemna i wywołane kolejne P,
	pierwsze w kolejności P oczekuje na V, a drugie i następne w kolejności oczekują na
	ten semafor

	N := 2; // inicjalizacja wartości semafora   // dlaczego N=2 ??? // tylko dla
przykladu
	oczekuje := false; // flaga oczekiwania na V przez P
end;
```

/////// inna wersja: //sprawdzicie?

```
var
   A,B : BINARY SEMAPHORE;
   x : condition; //może również być semaforem
   N : INTEGER;
```

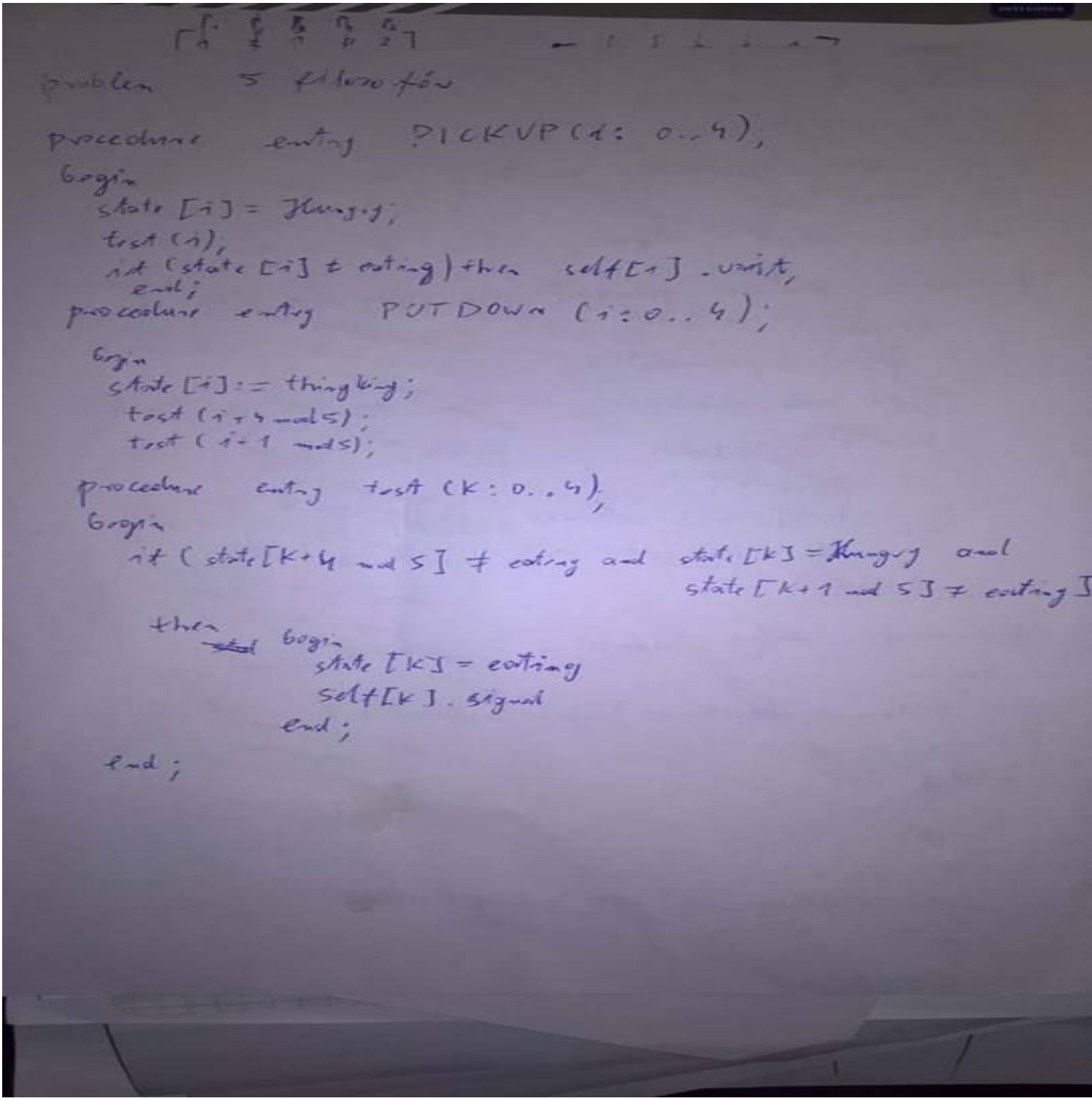
```
procedure P(x);
begin
    Pb(B);
    while (N - x) < 0 do
            x.wait;
    Pb(A);
    N := N - x;
    Vb(A);
    Vb(B);
end;

procedure V(x);
begin
    Pb(A);
    N := N + x;
    x.signal;
    Vb(A);
end;

begin
    A := true;
    B := true;
    N := 2;
end;
```

Przedstaw rozwiązanie problemu pięciu filozofów z użyciem mechanizmu wymiany komunikat

to było na wykładach? niedaleko stołu
bylo, ale nie z wymianą komunikatów, tylko jakis pseudokod
fakt

KTOS TO MA?

pseudokod z wykładu wymiana komunikatow -> operacje send(P, m) i receive(Q, m)
ten kod jest na monitorach zrobiony



Podaj warunki konieczne zakleszczenia.

- Wzajemne wykluczanie (dany zasób można przydzielić co najwyżej jednemu procesowi)
- Zachowywanie zasobu (proces oczekujący nie zwalnia swoich zasobów)
- Niewywłaszczalność (jedynie proces może zwolnić swoje zasoby
- istnienie cyklu oczekiwań (każdy proces ubiega się o dodatkowe zasoby)

Następujące rozwiązanie problemu sekcji krytycznej zostało zaprezentowane przez Hymana. Zbadaj, czy jest ono poprawne. Jeśli nie jest poprawne, pokaż przykład, który nie spełni jednego z trzech warunków sekcji krytycznej.
Dwa procesy, P_1 i P_2, dzielą następujące zmienne:

```
var flaga: array[1..2] of integer; (* początkowo 0 *)
numer: 1..2;
```

Poniższy program dotyczy procesu P_i; P_j oznacza drugi proces.

```
1.    repeat
2.    flaga[i]:=numer;
3.    while numer<>i do
4.    begin
5.          while flaga[j]<>0 do nic;
6.          numer:=i;
7.    end;
8.    ...
9.    sekcja krytyczna
10.   ...
11.   flaga[i]:=0;
12.   ...
13.   reszta
14.   ...
15.   until false;
```

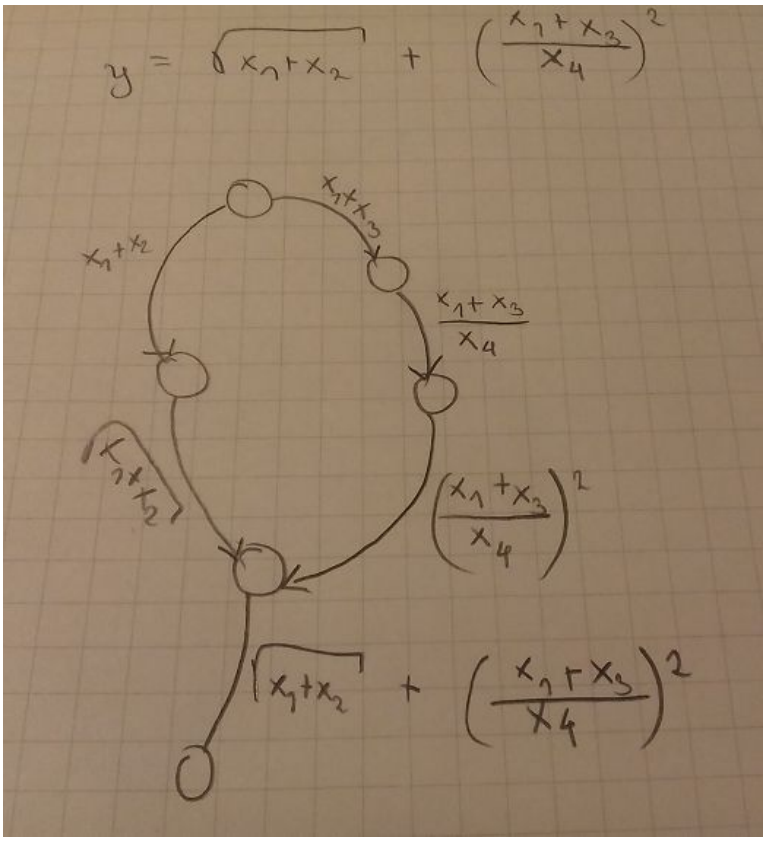
Silny warunek postepu nie jest spełniony?
proces wyjdzie z sekcji krytycznej, nie przełączając kontekstu, od razu bedzie chciał znow wejsc,do niej, przez co uniemozliwi drugiemu procesowi wejscie?
Na pewno nie jest spełniony warunek bezpieczeństwa. Dwa procesy moga wejść do sekcji krytycznej. Np. gdy P1 wyjdzie z pętli w 5 linii, nastąpi przełączenie kontekstu, P2 przejdzie przez pętlę w 3 linii bez wchodzenia do linii 4-7, to obydwa procesy będą w sekcji krytycznej, nawet jeśli zmienna numer przyjmie wartość i pierwszego procesu.

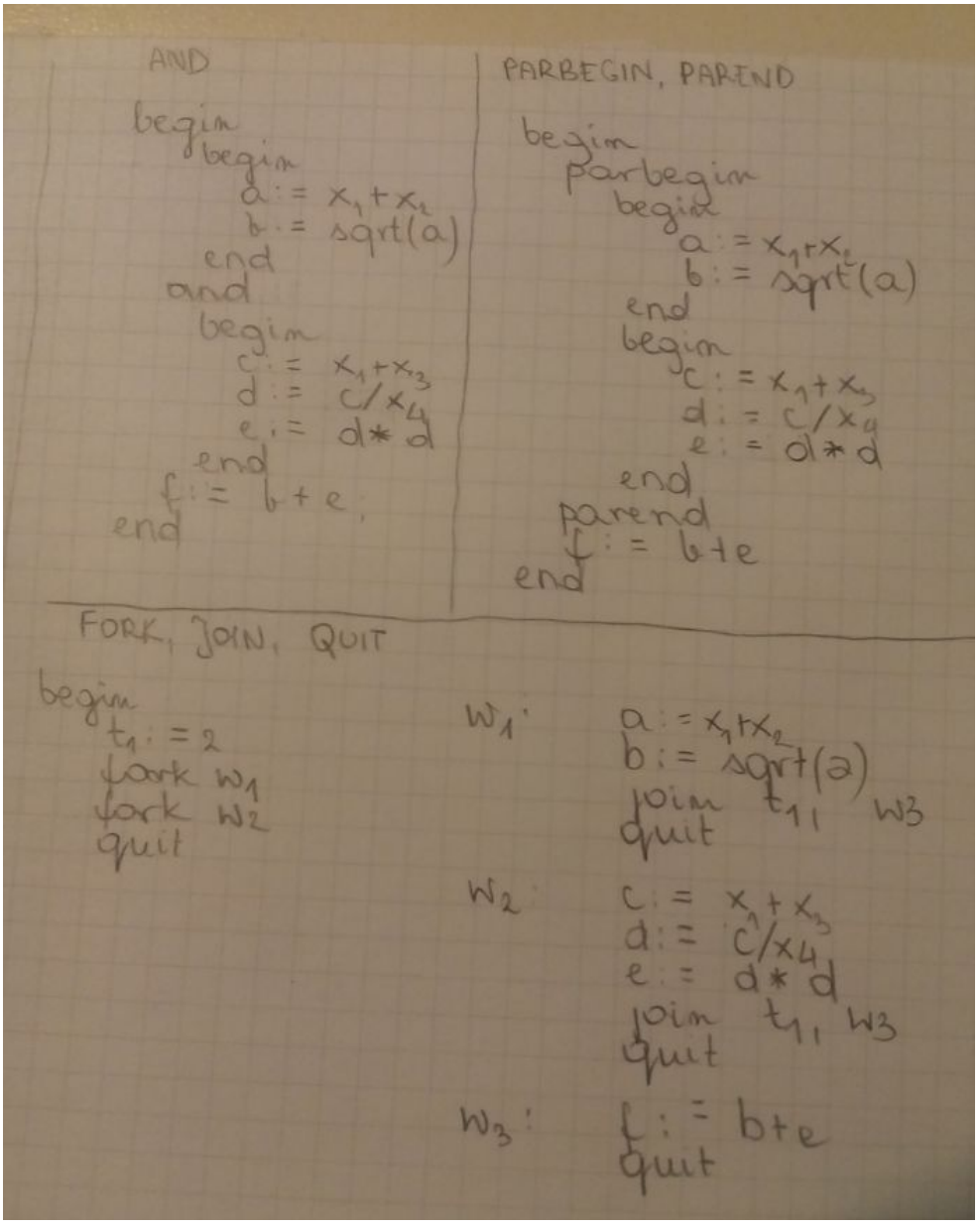
1. Procesy realizujące instrukcje poza sekcją krytyczną nie mogą uniemożliwiać innym procesom wejścia do sekcji krytycznej. Inaczej **słaby warunek postępu.**
2. Procesy powinny uzyskać dostęp do sekcji krytycznej w skończonym czasie. Inaczej **silny warunek postępu.**
3. Przy tych założeniach należy zagwarantować, że w każdej chwili czasu co najwyżej jeden proces jest w swej sekcji krytycznej. Nazywa sie to **warunkiem bezpieczeństwa.**

Przedstaw optymalny graf przepływu procesów oraz zgodne z nim programy współbieżne zaimplementowane z użyciem operacji „*parbegin-parend*” oraz „*fork-join-quit*”, odpowiadające współbieżnemu wyznaczaniu wartości następującego wyrażenia:

$$y = \sqrt{x_1 + x_2} + \left(\frac{x_1 + x_3}{x_4}\right)^2$$

//Poprawnie?
// wygląda w porządku, tylko brak średników w kodzie:-)

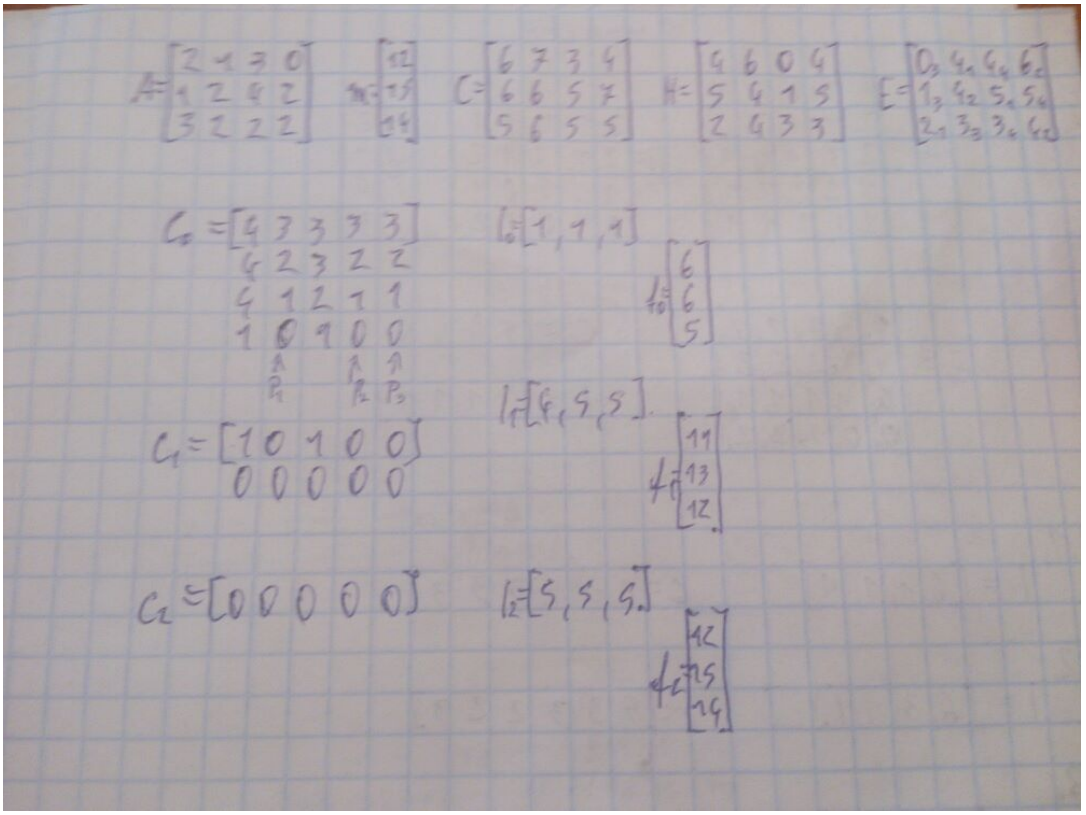



AND
begin
begin
a := x1 + x2
b := sqrt(a)
end
and
begin
c := x1 + x3
d := c / x4
e := d * d
end
f := b + e;
end
PARBEGIN, PAREND
begin
parbegin
begin
a := x1 + x2
b := sqrt(a)
end
begin
c := x1 + x3
d := c / x4
e := d * d
end
parend
f := b + e
end
FORK, JOIN, QUIT
begin
t1 := 2
fork W1
fork W2
quit
W1: a := x1 + x2
b := sqrt(a)
join t1, W3
quit
W2: c := x1 + x3
d := c / x4
e := d * d
join t1, W3
quit
W3: f := b + e
quit

Dany jest stan początkowy systemu:

$$A=\begin{bmatrix}2&1&3&0\\1&2&4&2\\3&2&2&2\end{bmatrix}\qquad m=\begin{bmatrix}12\\15\\14\end{bmatrix}\qquad C=\begin{bmatrix}6&7&3&4\\6&6&5&7\\5&6&5&5\end{bmatrix}$$

c) Stosując algorytm Holt'a sprawdź, czy stan jest bezpieczny.



d) Zakładając, że konsekwentnie stosowane jest podejście unikania, podaj sekwencję stanów systemu (zaznaczając procesy zawieszone) odpowiadającą następującego ciągowi żądań zasobów:

- w chwili $t_1 : p^a(P_1) = \begin{bmatrix}0\\1\\2\end{bmatrix}$
- w chwili $t_2 > t_1 : p^a(P_2) = \begin{bmatrix}2\\1\\1\end{bmatrix}$
- w chwili $t_3 > t_2 : p^r(P_4) = \begin{bmatrix}1\\1\\1\end{bmatrix}$

Należy przedstawić monitor synchronizujący współbieżnie generowane żądania drukowania. W rozwiązaniu należy przyjąć, że w pierwszej kolejności będą drukowane zadania krótkie. W tym celu należy wykorzystać operację *wait* priorytetową. W efekcie powinien być każdorazowo wyszukany proces oczekujący na wydruki i jego zadanie wydruku powinno być realizowane i następnie wyszukany następny proces jeśli takowy istnieje. (Operacja *wait* priorytetowa: *x.wait*(***priorytet***)).

Moja propzycja, jak kompletnie nietrafiona to piszcie

```
type KSERO = monitor
var x : condition
drukowanie : boolean

procedure chceDrukowac( int priorytet)
begin
        if drukowanie = true;
                x.wait(priorytet)
        drukowanie := true;
end

zwolnij zasoby
begin
        drukowanie :=false
        x.signal()
end

begin
        drukowanie = false;
end;
```

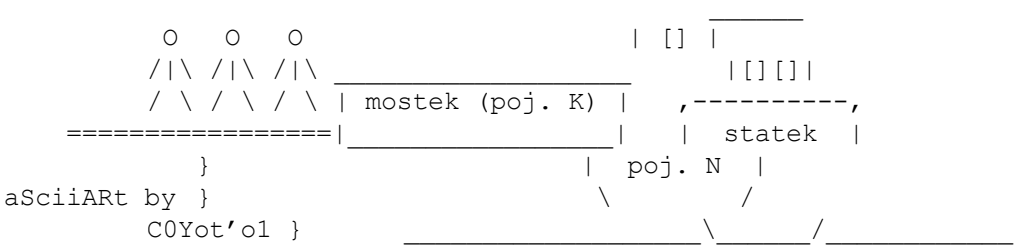
// wygląda ok, ktoś potwierdzi ?

// wersja alternatywna z buforem wydruku

```
N - rozmiar kolejki wydruku
M - ilosc priorytetow, liczone są od 0 w górę, gdzie 0 to najniższy

type KSERO = monitor
var kolejka, full : CONDITION;
      count, in, out, N = 10, M = 10, i : INTEGER;
      buffer : array[0..N] of TEXT;
      priorytet : array[0..N] of INTEGER;

procedure entry dodaj(dokument : TEXT, priorytet INTEGER)
begin
      if (count + 1) > N then full.wait;
      priorytet[in] := priorytet;
      buffer[in] := dokument;
      in := in + 1 mod N;
      count := count + 1;
      kolejka.wait(priorytet);
end;

procedure entry odbierzdrukuj
begin
      for i := M downto 0 do
      begin
            while count > 0 do
            begin
                  if priorytet[out] = i then
                  begin
                        priorytet[out] := -1;
                        drukuj(buffer[out]);
                        if count  = N then full.signal;
                        count := count - 1;
                        kolejka.signal;

                  end;
                  out := out + 1 mod N;
            end;
      end;
end;
```

```
procedure entry Init()
begin
      for i := 0 to N do priorytet := -1;
      count := 0;
      in := 0;
      out := 0;
end;
```

Przy nabrzeżu stoi statek o pojemności *N*. Statek z lądem jest połączony mostkiem o pojemności *K* (*K*<*N*). Na statek próbują dostać się pasażerowie, z tym, że na statek nie może ich wejść więcej niż *N*, a wchodząc na statek na mostku nie może być ich równocześnie więcej niż *K*. Statek co jedną godzinę wypływa w rejs. W momencie odpływania kapitan statku musi dopilnować aby na mostku nie było żadnego wchodzącego pasażera. Jednocześnie musi dopilnować by liczba pasażerów na statku nie przekroczyła *N*.
Napisać odpowiednio procedury *Pasażer* i *Kapitan* zsychnronizowane za pomocą jakiejkolwiek z metod synchronizacji procesów.

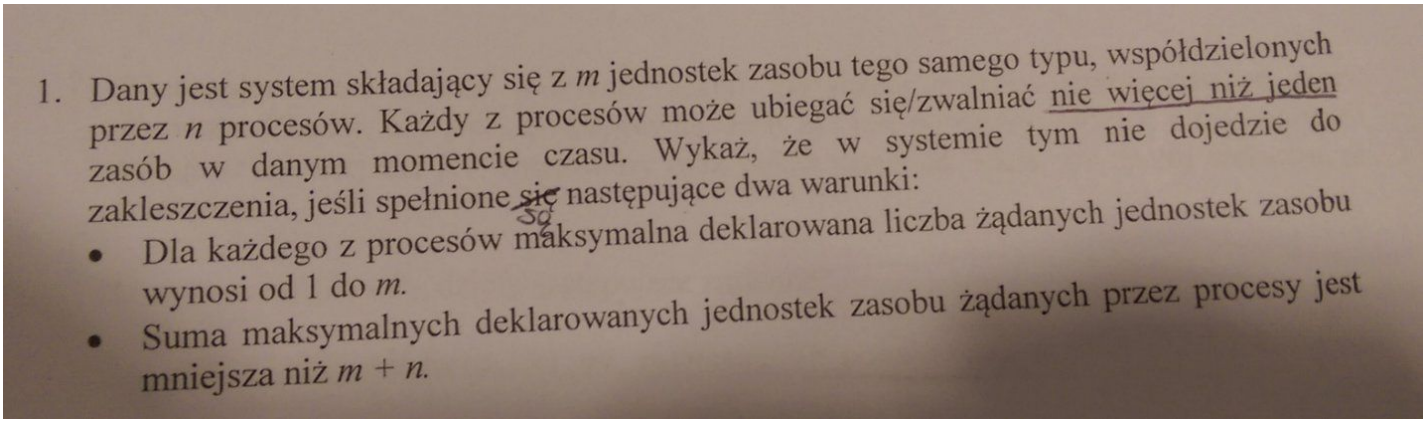



```
1  var
2      Ka : INTEGER;
3      aktywny : BOOLEAN;
4      mostek, statek : BINARY SEMAPHORE
5      //N - pojemność statku
6      //K - max ilość osób na mostku
7
8  procedure kapitan();
9  begin
10     while true do
11     begin
12         delay(3600);    //czekanie 1h
13         Pb(statek);
14         while Ka > 0 do
15             delay(random);  //czekanie na podróżnych
16
17         aktywny := true;
18         rejs();
19
20         Vb(aktywny);
21         Vb(statek);
22     end;
23 end;
24
25 begin
26     aktywny := false;
27     mostek := true;
28     statek := true;
29     Ka := 0;            //ilość osób na mostku
30 end;
```

```
1  procedure pasazer();
2  begin
3      while true do
4      begin
5          Pb(mostek);
6          Pb(statek);
7          if (N - 1 - Ka) < 0 then
8          begin
9              Vb(statek);
10             delay(random);
11             Pb(statek);
12         end;
13         Pb(na_mostku);
14         if (Ka+1) > K then
15         begin
16             Vb(na_mostku);
17             delay(random);
18             Pb(na_mostku);
19         end;
20         Ka := Ka + 1;
21         Vb(na_mostku);
22         Vb(statek);
23         Vb(mostek);
24
25         delay(10);              //przejście po mostku
26
27         Pb(na_mostku);
28         N := N - 1;             //wchodzenie na statek
29         Ka := Ka - 1;
30         Vb(na_mostku);
31
32         while aktywny = false do
33             delay(random);      //czekanie na rejs
34         w_podrozy();
35
36         Pb(statek);
37         N := N + 1;             //na ląd
38         Vb(statek);
39
40         delay(random);
41     end;
42 end;
```

ma ktoś coś lepszego/prostszego?

// ktos wie jak zrobic to ?



1. Dany jest system składający się z m jednostek zasobu tego samego typu, współdzielonych przez n procesów. Każdy z procesów może ubiegać się/zwalniać nie więcej niż jeden zasób w danym momencie czasu. Wykaż, że w systemie tym nie dojedzie do zakleszczenia, jeśli spełnione są następujące dwa warunki:
   - Dla każdego z procesów maksymalna deklarowana liczba żądanych jednostek zasobu wynosi od 1 do m.
   - Suma maksymalnych deklarowanych jednostek zasobu żądanych przez procesy jest mniejsza niż $m + n$.

taki luźny pomysł, bo nic innego do głowy nie przyszło (nie mam pojęcia czy prawidłowe):
skoro max żądanych zasobów to m+n-1 a każdy proces może zwolnić w jednej jednostce czasu maksymalnie 1 zasób to w 1 jednostce czasu może zostać zwolnionych n jednostek zasobu, więc żądanych będzie wtedy maksymalnie m-1, a mamy m jednostek zasobu więc nie nastąpi zakleszczenie.

Przy pomocy regionów krytycznych napisz rozwiązania problemu producenta-konsumenta

http://aragorn.pb.bialystok.pl/~wkwedlo/OS1-5.pdf

```
var buffer: shared record
      buffer: array [0..n-1] of ITEM;
      count, in, out: INTEGER;
end;

procedure PRODUCER(nextp: ITEM)
begin
      region buffer when count < n
      do
      begin
            buffer[in] := nextp;
            in:= in+1 mod n;
            count := count + 1;
      end;
end;

procedure CONSUMER(nextc: ITEM)
begin
      region buffer when count > 0
      do
      begin
            nextc := buffer[out];
            out := out+1 mod n;
            count := count - 1;
      end;
end;

begin
      count := 0;
      in := 0;
      out := 0;
end;

// lub coś takiego

var buffer: shared record
      buffer: array [0..n-1] of ITEM;
      count, in, out: INTEGER;
end;

procedure PRODUCER(nextp: ITEM)
begin
      region buffer do
      begin
            await(count < n);
            buffer[in] := nextp;
            in:= in+1 mod n;
            count := count + 1;
      end;
end;

procedure CONSUMER(nextc: ITEM)
begin
      region buffer do
      begin
            await(count > 0);
            nextc := buffer[out];
            out := out+1 mod n;
            count := count - 1;
      end;
```

```
end;

begin
	count := 0;
	in := 0;
	out := 0;
end;
```

Jest algorytm używania dysku względem kojenych żądań dostępu. Polega na tym że kolejne żądania są szeregowane, a głowica realizuje je w zależności od kierunku w którym zmierza. Używając monitorów napisz funkcje request(dest) i zwolnij() (po angielsku jakoś tak) dla użycia dysku.

W pewnym systemie operacyjnym plik tekstowy jest współdzielony przez współbieżnie działające procesy, posiadające jednoznaczne numery identyfikacyjne będące wartościami naturalnymi. W danym momencie czasu dostęp do pliku może być jednocześnie przydzielony pewnej liczbie procesów, dla których spełniony jest następujący warunek: suma wszystkich jednoznacznych numerów identyfikacyjnych przypisanych tym procesom jest mniejsza od wartości N. Zaproponuj implementację monitora koordynującego dostęp do pliku.

```
var count, N: INTEGER;
x: CONDITION;

procedure entry wejscie(id: INTEGER);
begin
        while (count + id) >= N do x.wait;
        count := count+id;
end;
procedure entry wyjscie(id: INTEGER);
begin
        count := count-id;
        x.signal;
end.

begin
        count := 0;
        N := 10;  // ograniczenie

end.
```

// ja bym to taki zapisał:

```
var count, N, stop: INTEGER;
      x : CONDITION;

procedure entry wejscie(id: INTEGER);
begin

      if (count + id) >= N then
      begin
            stop := stop + 1;
            x.wait;
      end;
      else count := count+id;
end;

procedure entry wyjscie(id: INTEGER);
begin
      count := count-id;
      if stop > 0 then
      begin
            x.signal;
            stop := stop - 1;
      end;
end;

procedure entry Init
begin
      count := 0;
      stop := 0;
      N := 10;  // ograniczenie

end;
```

Wykaż poprzez kontrprzykład (podaj odpowiedni przeplot operacji), że poniższa implementacja operacji semaforowych jest niepoprawna

```
procedure P(var s: Semaphore)
  begin
        L: if s > 0 then
        s := s-1;
        else goto L;
  end;

procedure V(var s: Semaphore)
  begin
        s := s+1;
  end;
```

Tutaj był maks za znalezienie jednego z możliwych syfów jakie mogą zajść. Najprościej:

P1, P2 - procesy próbujące wykonać operację P

Fatalny przeplot, na początku s = 1:
P1: if s > 0 then - po tej operacji P1 wykona operację dekrementacji, ponieważ s jest na razie równe 1, lecz zanim to nastąpi kontekst jest przełączany na P2
P2: if s > 0 then - analogicznie jak dla P1 s jest jeszcze równe 1, więc następuje przejście do dekrementacji
P2: s := s - 1; - teraz kontekst wraca do P1
P1: s := s - 1;

Następnie oba procesy kończą operację P i żaden z nich nie wykona instrukcji
else goto L
która jest pętlą oczekiwania. Więc jeśli semafor kontrolował dostęp do sekcji krytycznej, to oba procesy wejdą do niej.

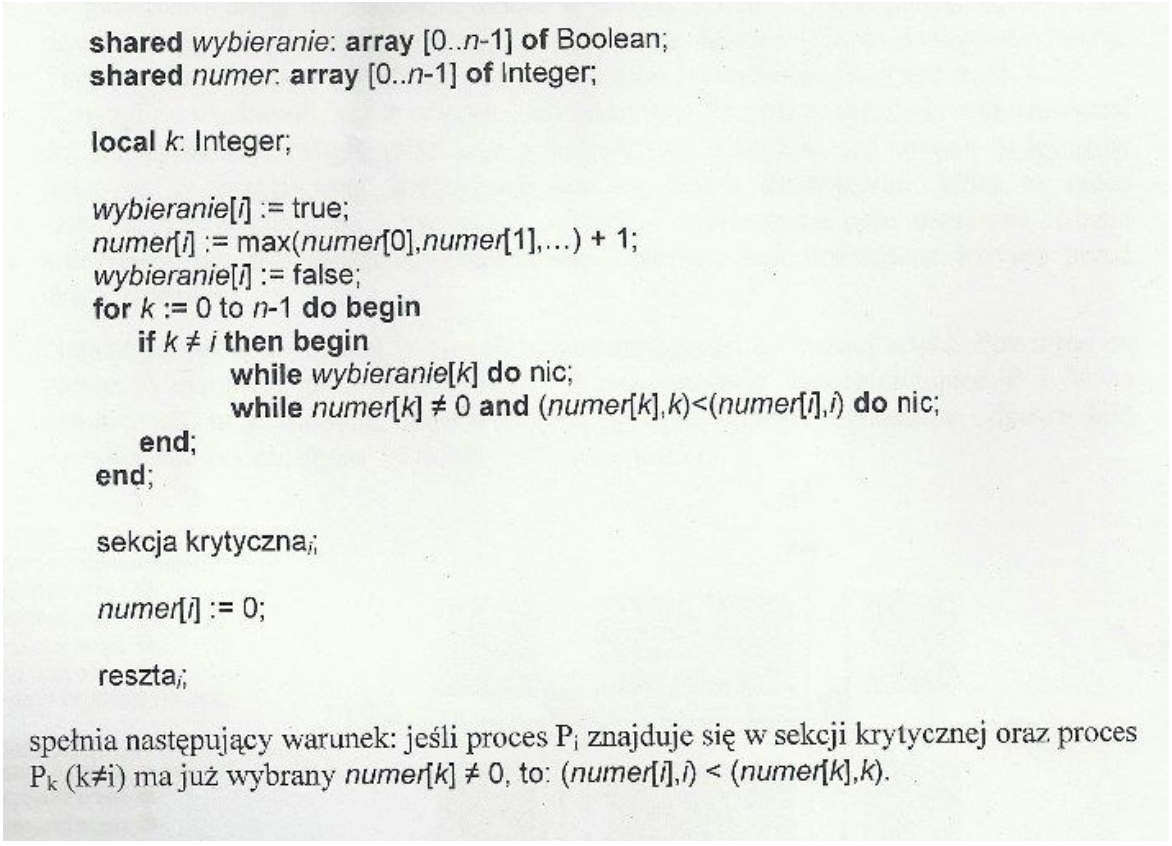
Można ew. jeszcze się trochę rozpisywać, bo Stroiński obcinał punkty jak nie wiedział o co chodzi, ale jak tylko się poszło i kłóciło 10 min i wychodziło, że się wie o co chodzi, to podwyższał. Uznawane było również rozwiązanie, w którym zauważyło się nieatomowość operacji s := s - 1; dla P oraz s := s + 1; Dla nich łatwo znaleźć przeplot, który coś psuje.

Suma $x_1+x_2+x_3+...+x_{n-1}+x_n$ i zadaniem jest zbudować optymalny graf przepływu procesów, a następnie to zakodzić w notacji parbegin-parend, and oraz fork-join-quit. N to dowolna liczba naturalna
Rozwiązaniem jest funkcja rekurencyjna, która jest bardzo podobna dla wszystkich reprezentacji. Nie pamiętam dokładnie jak wyglądało rozwiązanie, które mi pokazał sprawdzający, ale coś, co by na pewno zaakceptowali wygląda tak:

```
// a, b oznaczają zakresy w tablicy tab, w których sumujemy
add(Array tab, int a, int b){
	if(b - a < 3){
		tab[a] += tab[a + 1];
		if(b - a == 2){
			tab[a] += tab[b];
		}
	}else{
		h = floor((a + b) / 2); // w pseudokodzie podłoga/sufit przy dzieleniu nawet całkowitych to coś co warto zaznaczyć

		parbegin
			add(tab, a, h);
			add(tab, h + 1, b);
		parend
		tab[a] += tab[h + 1];
	}
}
////////////////////////////////////////
Notacja and różni się wyłącznie tym, że ma

add(tab, a, h);
	and
add(tab, h + 1, b);

zamiast parbegin, parend


Dla notacji fork-join-quit zmieni się trochę więcej:

add(Array tab, int a, int b){
	if(b - a < 3){
		tab[a] += tab[a + 1];
		if(b - a == 2){
			tab[a] += tab[b];
		}
	}else{
		int t = 2;
		int h = floor((a + b) / 2);

		fork(S1);
		fork(S2);
		quit;
		S1: add(tab, a, h);
				join t, R;
				quit;
		S2: add(tab, h + 1, b);
				join t, R;
				quit;
		R: tab[a] += tab[h + 1];
	}
}
```

Taka funkcja sumuje w miejscu - w samej tablicy, wynik daje w tab[0]. Nieparzystość jest rozwiązywana dla warunku brzegowego rekurencji. Proponuję sobie przetestować na kartce dla 10 wartości, żeby zaczaić. Jak komuś bardziej pasuje Pascal, to sobie może to pisać inaczej, ale mówili, że może być pseudo-C.

Algorytm Lamporta "piekarni" (na wzajemne wykluczanie się n procesów) - wykazać, że spełnia następujący warunek: jeśli proces Pi znajduje się w sekcji krytycznej oraz proces Pk (k!=i) ma już wybrany numer[k]!=0, to: (numer[i],i) < (numer[k],k)



```
shared wybieranie: array [0..n-1] of Boolean;
shared numer: array [0..n-1] of Integer;

local k: Integer;

wybieranie[i] := true;
numer[i] := max(numer[0],numer[1],...) + 1;
wybieranie[i] := false;
for k := 0 to n-1 do begin
    if k ≠ i then begin
            while wybieranie[k] do nic;
            while numer[k] ≠ 0 and (numer[k],k)<(numer[i],i) do nic;
    end;
end;

sekcja krytyczna_i;

numer[i] := 0;

reszta_i;
```

spełnia następujący warunek: jeśli proces P_i znajduje się w sekcji krytycznej oraz proces P_k (k≠i) ma już wybrany *numer*[k] ≠ 0, to: (*numer*[i],i) < (*numer*[k],k).

Pierwsza rzecz w takim zadaniu to oczywiście ponumerowac sobie linijki, wtedy można o czymś rozmawiać. Trzeba zauważyć, że mamy dwa przypadki do rozważenia.
1. Pierwszy to taki, w którym proces i-ty szybciej przeszedł fazę wybierania numerka, niż k-ty dojechał do i-tego wyrazu szukając maksimum. Dalej trzeba wspomnieć o tym, że dzięki temu, że wybieranie jest zabezpieczone przez zmienną, która wraz z pierwszą pustą pętlą gwarantuje, że żaden numerek nie będzie odczytany w czasie wybierania i proces nie wyskoczy zaraz z czymś innym po tym jak już nasz proces się z nim porównał. To wszystko sprawi, że numer[i] będzie ostro większy od numer[k], więc proces i-ty przejdzie do sekcji krytycznej, natomiast proces k-ty być może zawiesi się na drugiej pętli, ale z całą pewnością w tym samym momencie nie wejdzie do sekcji krytycznej.
2. W mniej oczywistym scenariuszu proces i-ty sprawdzi co najmniej k pierwszych numerów, a k-ty co najmniej i pierwszych numerów zanim jego rywal ustali swój numer. Wtedy może być taka sytuacja, że numer[i] == numer[k]. Tutaj kluczowe jest to, że zabezpieczone jest to wybieranie, ponieważ procesy wzajemnie nie sprawdzą warunku przejścia do sekcji krytycznej zanim ten drugi nie wybierze tego samego numerku. Wtedy jeżeli wiemy, że proces i-ty przeszedł do sekcji krytycznej, to wiemy, że i musi być mniejsze od k. Więc warunek drugiej pętli będzie spełniony wyłącznie dla k-tego i on się zatrzyma, natomiast i-ty przejdzie dalej.
Każdy inny scenariusz sprowadza się do jednego z powyższych. W każdym z nich albo numer[i] < numer[k] bądź numer[i] == numer[k], lecz i < k. Jest to warunek zapewniający bezpieczeństwo.
C.N.D.

Dla 10 punktów wprowadzić na wszelki wypadek numerację linii i jak o czymś się mówi to korzystać z numerów. I ładniej to ująć i rozwinąć wszystkie skróty myślowe.

Zaimplementuj P i V za pomocą operacji TestAndSet.
// materiał z wykładu Synchronizacja Cz. 3

```
program PV_IMPLEMENTATION;
      var active, delay: BOOLEAN;
      var NS: INTEGER;

procedure PIMPLEMENTATION;
var pActive : BOOLEAN;
begin
      Disable interrupts;
      pActive:=True;
      while pActive do
            testandset(pActive,active);
            NS:=NS-1;
            if NS >= 0 then
            begin
                  S:=S-1;
                  active:=False;
                  Enable interrupts;
            end
            else
            begin
                  Block process invoking P(S);
                  p:= Remove from RL;
                  active:=False;
                  Transfer control to p with
                  Enable interrupts;
             end;
 end;

procedure VIMPLEMENTATION;
 var vActive : BOOLEAN;
 begin
      Disable interrupts;
      vActive:=True;
      while vActive do
               testandset(vActive,active);
            NS:=NS+1;
            if NS > 0 then
                  S:=S+1;
            else
              begin
                  p:=remove from LS;
                  Add p to RL;
              end;
            active:=False;
            Enable interrupts;
 end;
```

Legenda:
LS - List associated with S
RL - Ready List

Algorytm Lamporta "piekarni" ze skreślonym choosingiem. Napisać przeplot operacji taki, że wynik będzie nieprawidłowy.

Algorytm Lamporta dla *n* procesów

```
1.  program LAMPORTALGORITHM;
2.  begin
3.   shared
4.     choosing[1..n]: 0..1;
5.     num[1..n]: INTEGER;
6.   local
7.     test_i: 0..1;
8.     k, mine_i : INTEGER;
9.     other_i, temp_i: INTEGER;
10. while True do
11.  begin
12.    choosing[i]:=1;
13.    mine_i:= 0;
14.    for k:=1 to n do
15.      if k≠i then
16.       begin
17.         temp_i:=num[k] ;
18.         mine_i:=max(mine_i,temp_i);
19.       end;
20.    mine_i:= mine_i+1;
21.    num[i]:= mine_i;
22.    choosing[i]:=0;
23.    for k:=1 to n do
24.        if k≠i then
25.         begin
26.           repeat
27.             test_i:=choosing[k];
28.           until test_i=0;
29.           repeat
30.             other_i:=num[k];
31.           until other_i=0 or
               (mine_i,i)<(other_i,k);
32.         end;
33.     criticalSection;
34.     num[i]:=0 ;
35.     reminderSection;
36.   end;
37. end.
```

(c) Zakład Systemów Informatycznych

Slajd 15

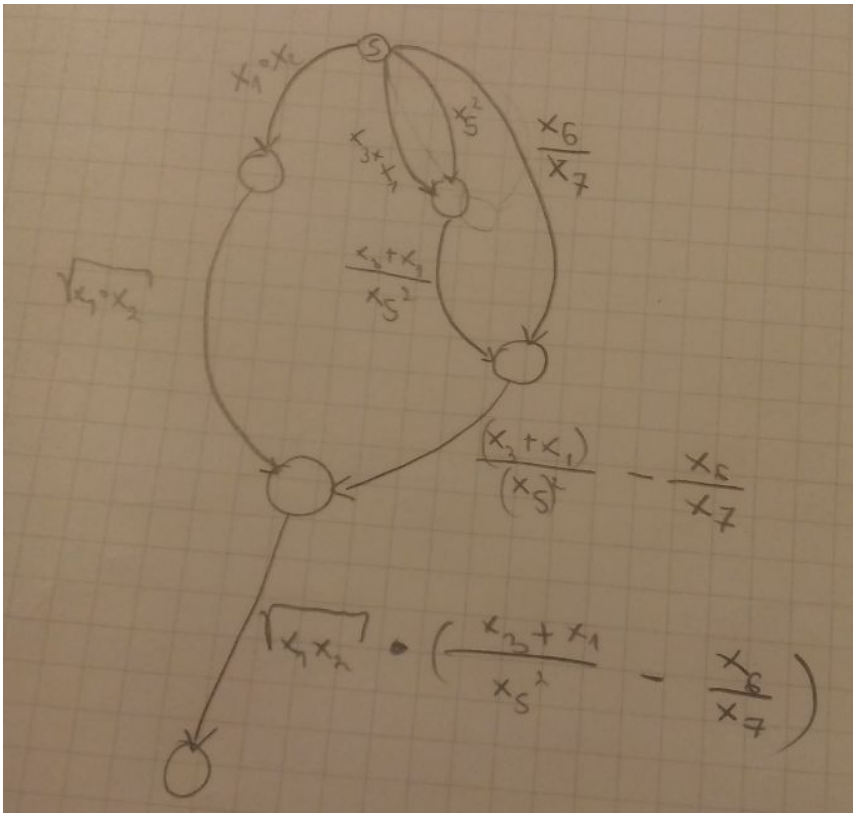
Załóżmy że w systemie są dwa procesy A i B o indeksach Ai < Bi które wykonują się współbieżnie do linijki 20 włącznie. Wtedy rusza proces B i jako że ma najmniejszy numer (proces A nie przypisał jeszcze wartości mine do zmiennej globalnej num) wchodzi do sekcji krytycznej. Wtedy budzi się proces A który ma ten sam numer co B (zatem decyduje indeks), a indeks Ai < Bi więc beztrosko wchodzi do zajętej już sekcji krytycznej.

warunek bezpieczeństwa nie jest spełniony

//linijka (mine,i) < (other,k) oznacza tyle co: if(mine==other) return i<k; else return mine<other;

flaga choosing blokuje w tym przypadku wejście do sekcji krytycznej procesu B aż proces A zapisze swój numer do zmiennej globalnej num[].

Graf, fork join quit, notacja and oraz parbegin parend równania y=sqrt(x1*x2) *((x3+x1)/x5^2 - x6/x7)
//Poprawne? - tak



```
begin
    begin
        y1:= x1*x2;
        y2:= sqrt(y1);
    end
    and
    begin
        begin
            begin
                y3:= x3+x1;
            end
            and
            begin
                y4:= x5*x5;
            end
            y5:= y3/y4;
        end
        and
        begin
            y6:= x6/x7;
        end
        y7:= y5 - y6;
    end
    y:=y2*y7;
end
```

```
w1: y1:= x1*x2;
    y2:= sqrt(y1);
    join t3,w7 ;
    quit;

w2: y3:= x3+x1;
    join t1, w4;
    quit;

w3: y4:= x5*x5;
    join t1, w4;
    quit;

w4: y5:= y3/y4;
    join t2, w6;
    quit;

w5: y6:= x6/x7;
    join t2, w6;
    quit;

w6: y7:= y5-y6;
    join t3, w7;
    quit;

w7: y:= y2*y7;
    quit;

begin
    t1=2;
    t2=2;
    t3=2;

    fork w1;
    fork w2;
    fork w3;
    fork w4;
    quit;
end
```

```
begin
    parbegin
        begin
            y1:= x1*x2;
            y2:= sqrt(y1);
        end
        begin
            parbegin
                begin
                    parbegin
                        y3:= x3+x1;
                        y4:= x5*x5;
                    parend
                    y5:= y3/y4;
                end
                begin
                    y6:= x6/x7;
                end
            parend
            y7:= y5 - y6;
        end
    parend
    y:= y2 * y7;
end
```

sformułować i udowodnić warunek postępu w algorytmie Lamporta

Algorytm Lamporta dla *n* procesów

```
1.  program LAMPORTALGORITHM;
2.  begin
3.   shared
4.     choosing[1..n]: 0..1;
5.     num[1..n]: INTEGER;
6.   local
7.     test_i: 0..1;
8.     k, mine_i : INTEGER;
9.     other_i, temp_i: INTEGER;
10. while True do
11.   begin
12.     choosing[i]:=1;
13.     mine_i:= 0;
14.     for k:=1 to n do
15.       if k≠i then
16.         begin
17.           temp_i:=num[k] ;
18.           mine_i:=max(mine_i,temp_i);
19.         end;
20.     mine_i:= mine_i+1;
21.     num[i]:= mine_i;
22.     choosing[i]:=0;
23.     for k:=1 to n do
24.         if k≠i then
25.          begin
26.            repeat
27.              test_i:=choosing[k];
28.            until test_i=0;
29.            repeat
30.              other_i:=num[k];
31.            until other_i=0 or
                 (mine_i,i)<(other_i,k);
32.          end;
33.      criticalSection;
34.      num[i]:=0 ;
35.      reminderSection;
36.    end;
37. end.
```

(c) Zakład Systemów Informatycznych

Slajd 15

Procesy realizujące instrukcje poza sekcją krytyczną nie mogą uniemożliwiać innym procesom wejścia do sekcji krytycznej. Inaczej **słaby warunek postępu.**

Procesy powinny uzyskać dostęp do sekcji krytycznej w skończonym czasie. Inaczej **silny warunek postępu.**

Dla procesu P_i kod wygląda następująco:

```
1 number : integer;
2 repeat
3 number := i;
4 until number = i;
5 CRITICAL_SECTION;
```

Podać takie przebiegi procesów, gdzie a) nastąpi złamanie warunku bezpieczeństwa, b) nie nastąpi złamanie warunku bezpieczeństwa

a) Proces pierwszy wchodzi do pętli w linii 2, zmienia numer na swój, wychodzi z pętli (linia 4), następuje przełączenie kontekstu, proces drugi wchodzi do pętli (2), zmienia numer na swój, wychodzi z pętli (4) -> oba procesy mogą wejść do sekcji krytycznej
b) Przełączenie kontekstu, które wyżej wystąpiło po wykonaniu przez proces 1. linii nr 4, mogłoby nastąpić po wykonaniu linii 3 i proces nr 1. byłby poza sekcją krytyczną, pod warunkiem, że proces 2. zdołałby wykonać swoje procedury w sekcji krytycznej przed ponowną iteracją pętli dla procesu 1.

//Coś za proste to zadanie chyba

Algorytm H... Sprawdzic czy dziala poprawnie. Jesli nie to jaki warunek nie jest spelniony? Podac przyklad. Mamy dwa procesy.

```
flag: array of int [1..2] /* poczatkowo 0 */
dane: 1..2
repeat
  flag[i]:=numer;
  while numer <> i do
  begin
          while flag[j] <> 0 do nic;
          numer:=i;
  end;
  sekcja_krytyczna
  flag[i]=0
until false;
```

Semafory. W ZOO jest slon, zyrafa, lew, tygrys i malpa. Zwierzeta wypuszczane sa na wybieg. Jednoczesnie moga byc trzy zwierzatka, przy czym jesli jest slon to tylko dwa.

slon nie moze byc z lwem
zyrafa nie moze byc z lwem i tygrysem
malpa i tygrys musza byc jednoczesnie
lew nie moze byc z tygrysem

Zadanie praktycznie takie samo jak to z remontem
inicjalizacja:
I(zw,3)
I(SL,1)
I(ŻLT,1)
I(M,0)
I(T,1)
ziw

Słoń	Żyrafa	Lew	Tygrys	Małpa
			P(zw,2)	
			P(ŻLT)	
P(SL)	P(ŻLT)	P(ŻLT)	P(T) V(M)	
P(zw,2)	P(zw)	P(zw)		P(M) V(T)
wybieg	wybieg	wybieg	wybieg	wybieg
V(zw,2)	V(zw)	V(zw)	V(zw,2)	
V(SL)	V(ŻLT)	V(ŻLT)	V(ŻLT)	

słoń	żyrafa	lew	tygrys	małpa
P(R,1)	P(R,1)	P(R, 2)	P(R,2)	
P(ZW,2)	P(ZW,1)	P(ZW,1)	P(ZW,1)	P(ZW,1)
			V(M)	V(T)
			P(T)	P(M)
wybieg	wybieg	wybieg	wybieg	wybieg
			V(M)	V(T)
			P(T)	P(M)
V(ZW,2)	V(ZW,1)	V(ZW,1)	V(ZW,1)	V(ZW,1)
V(R,1)	V(R,4)	V(R,5)	V(R,2)	

przy 4 semaforach: R=2 ; ZW = 2 ; T=M=0

UWAGA! żyrafa: V(R,1) oraz lew: V(R,2)
oczywiście semafor ZW można wywalić

słoń:	Zyrafa:	Lew	tygrys	Malpa
----------	------	------	-----	-------
P(SLT,1)	P(ZL,1)	P(ZL,1)	P(SLT,1)	P(ZL,1)
P(zwierz,2)	P(ZLT,1)	P(SLT,1)	P(ZLT,1)	P(zwierz,2)
	P(zwierz,1)	P(ZLT,1)	P(zwierz,1)	
		P(zwierz,1)		
wybieg	wybieg	wybieg	wybieg	wybieg
V(zwierz,2)	V(zwierz,1)	V(zwierz,1)	V(zwierz,1)	V(zwierz,2)
V(SLT,1)	V(ZLT,1)	V(ZLT,1)	V(ZLT,1)	V(ZL,1)
	V(ZL,1)	V(SLT,1)	V(SLT,1)	
		V(ZL,1)		

Algorytm Petersona dla 3 przykladow: a) petla for k:=1 to n-2 b) petla for k:=1 to n-1 c) petla for k:=1 to n

Analiza poprawnosci.

Dla a) algorytm niepoprawny (w najgorszym przypadku do sekcji krytycznej moga wejsc 2 procesy) dla b) algorytm poprawny (jest to normalna wersja tego algorytmu) dla c) algorytm poprawny (ale ostatnia: n-ta iteracja jest nadmiarowa, gdyz wykona ja i tak tylko 1 proces)

Przedstaw implementację wielowartościowych operacji semaforowych P(S,n), V(S,n) oraz operacji inicjującej I(S,n) z użyciem binarnych operacji semaforowych Pb(B) i Vb(B).
Rozwiązanie

```
/* Moja propozycja... by Materac
 */

void I(unsigned int S, unsigned int n) {
  B = 1;
  B2= 1;
  S = n;
}

void V(unsigned int S, unsigned int n) {
  Pb(B);
  S += n;
  Vb(B2);
  Vb(B);
}

void P(unsigned int S, unsigned int n) {
  Pb(B);
  Pb(B2);
  while (S<n) {
          Vb(B);
          Pb(B2);
          Pb(B);
  }
  S -= n;
  Vb(B2);
  Vb(B);
}
////////////////////////////////////////////////
```

Wersja w której N może spaść poniżej 0, ale blokujemy wykonywanie P gdy tak się stanie:

```
P(SO)
{
          Pb(S); // sekcja krytyczna dostępu do "N"
          N--;
          if (N<0)
          {
          Vb(S); // już nie potrzebujemy dostępu do N
          Pb(D); // D da się opuścić tylko gdy jakieś V go podniesie
          }
          else
          Vb(S);
}

************************
V(SO)
{
          Pb(S);
          N++;
          if (N<=0)
          Vb(D); // odblokuj proces który utknął na <=0
```

```
        Vb(S);
}
************************
S=1, D=0;
```

Wersja w której N nie może spaść poniżej zera:

```
P(SO)
{
        Pb(F); // sekcja krytyczna semafora P - żaden inny niech się równolegle nie wykonuje!
        Pb(S); // sekcja krytyczna dostępu do "N"
        if (N==0)
        {
        Vb(S); // już nie potrzebujemy dostępu do N
        Pb(D); // D da się opuścić tylko gdy jakieś V go podniesie;
        Pb(S);
        }
        N--;
        Vb(S);
        Vb(F);
}

************************
V(SO)
{
        Pb(S);
        N++;
        Vb(D); // odblokuj proces który utknął na ==0
        Vb(S);
}
************************
S=1, D=0; F=1
```

Zapisz przy pomocy monitora problem producent-konsument

Problem producent-konsument z wykorzystaniem monitora

```
1. type PRODUCER_CONSUMER = monitor
2. var full, empty : CONDITION;
3. count : INTEGER;

4. procedure entry ENTER;
5. begin
6. if count = N then full.wait;
7. enter_item;
8. count := count + 1;
9. if count = 1 then empty.signal;
10. end;

11. procedure entry REMOVE;
12. begin
13. if count = 0 then empty.wait;
14. remove_item;
15. count := count - 1;
16. if count=N - 1 then full.signal;
17. end;
21. procedure PRODUCER;
22. begin
23. while True do
24. begin
25. produce_item;
26. PRODUCER_CONSUMER.enter;
27. end
28. end;

29. procedure CONSUMER;
30. begin
31. while True do
32. begin
33. PRODUCER_CONSUMER.REMOVE;
34. consume_item
35. end;
36. end.
```

1. semafor dwustronnie ograniczony - nie moze przekroczyc wartosci smax:smax>0 dla semafora zdefiniowano operacje P(s) : czekaj az s>0 i wtedy s:=s-1; V(s) : czekaj az s<smax i wtedy s:=s+1; zaproponuj implementacje monitora realizujacego takie operacje semaforowe

```
type PV = monitor;
var maxok,minok : condition;
int s,smax;
begin

  procedure entry Pproc()
  begin
        if s<=0 then minok.wait ;
        s:=s-1;
        maxok.signal;
  end;

  procedure entry Vproc()
  begin
        if s>=smax then maxok.wait ;
        s:=s+1;
        minok.signal;
  end;

  begin
  s:=WARTOSC_POCZATKOWA //lub 0
  smax:=OGRANICZENIE_GORNE
  end;

end;
```

Zaproponuj rozwiązanie problemu czytelników i pisarzy za pomocą mechanizmu monitora, przy założeniu, że priorytet mają pisarze. Oznacza to, że jeśli pisarz czeka na wejście do czytelni zajętej przez czytelników, to nie wpuszcza już nowych czytelników.

```
class ReadersWriters : public Monitor {
        bool writer;
        int nr;
        Condition readers, writers;

public:
 entry void startRead() {
        if (writer) {
        readers.wait();
        }
        if (!writer) {
        readers.signal();
        }
        nr++;
 }

 entry void endRead() {
        nr--;
        if (nr == 0) {
        writers.signal();
        }
 }

 entry void startWrite() {
        if (writer) {
        writers.wait();
        } else if (nr > 0) {
        writer = true;
        writers.wait();
        }
        writer = true;
 }

 entry void endWrite() {
        writer = false;
        if (writers.queue()) {
        writers.signal();
        } else if (readers.queue()) {
        readers.signal();
        }
 }

 ReadersWriters() {
        writer = false;
        nr = 0;
 }
```

---

3. Rozważmy system złożony z procesów P_1, P_2, ..., P_n, z których każdy ma przypisany unikalny priorytet. Napisz monitor, zarządzający trzema jednakowymi drukarkami, który przydziela wolną drukarkę w kolejności zgodnej z priorytetem procesu.

```
type resource = monitor
var P: array[0..2] of boolean;
X: condition;
procedure acquire (id: integer, printer-id: integer);
   begin
      if P[0] and P[1] and P[2] then X.wait(id)
      if not P[0] then printer-id := 0;
            else if not P[1] then printer-id := 1;
                  else printer-id := 2;
      P[printer-id]:=true;
   end

procedure release (printer-id: integer)
  begin
      P[printer-id]:=false;
      X.signal;
   end

   begin
      P[0] := P[1] := P[2] := false;
end
```

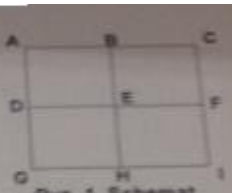
// rozwiazanie znalezione w necie, czy ktos wie gdzie tutaj wykorzystują priorytet procesu ? bo ja nie widze nie ma, bardziej to jest poprawne:

```
type PrinterMonitor = monitor

var
   printerTaken: array 0..2 of boolean; // initially false
   max: integer;
   x: condition;

function entry askForPrinter(priority: integer): integer //assuming that higher number equals higher priority and that priority is positive

var
   i: integer;

begin
   repeat
      i := 0;
      while i < 3 do
      begin
         if not printerTaken[i] then
            break;
         Inc(i);
      end
      if i = 3 then
      begin
         repeat
```

```
            x.wait();
            if priority > max then
                max := priority;
            x.signal();
        until priority = max;
    end
  until i <> 3;
  printerTaken[i] := true;
  Result := i;
end

procedure entry releasePrinter(printerIndex: integer)
begin
  printerTaken[printerIndex] := false;
  max = 0;
  x.signal();
end

procedure main(priority: integer)
var
  printerIndex: integer;
begin
  printerIndex := askForPrinter(priority);
  print(printerIndex);
  releasePrinter(printerIndex);
end
```

Załóżmy, że udostępniane przez monitor operacje wait(c) i signal(c) na zmiennej warunkowej c zastąpimy pojedynczą konstrukcją await(B) (podobnie jak w warunkowych regionach krytycznych), gdzie B jest ogólnym wyrażeniem boolowskim. Operacja await powoduje, że działanie wykonującego ją procesu jest wstrzymane do chwili gdy spełniony zostanie warunek B. Napisz przy użyciu tej konstrukcji monitor implementujący problem czytelników i pisarzy, zakładając, że priorytet posiadają procesy pisarzy.

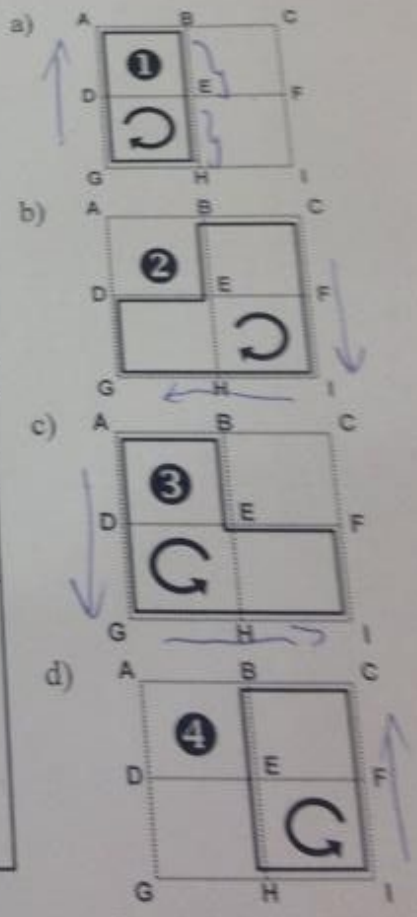
4. W mieście Speedvillage radcy miejscy postanowili zmodernizować komunikację miejską. Zgodnie z uchwałą rady wybudowano nowoczesną sieć kolei magnetycznej łączącą 9 przystanków oznaczonych kolejnymi literami alfabetu A,B,...,H,I w sposób pokazany na rysunku 1. Zakupiono też 4 nowoczesne pociągi, które mają obsługiwać 4 linie oznaczone numerami 1, 2, 3 i 4. Przebieg i kierunek poszczególnych tras został pokazany na rysunkach 2a-d. Z uwagi na ograniczenia technologiczne pociągi nie mogą poruszać się do tyłu. Nowoczesne systemy ostrzegania zamontowane w tych pociągach uniemożliwiają zderzenie się pojazdów poruszających się w tym samym kierunku.



Rys. 1 Schemat połączeń

Zaprojektuj system synchronizacji ruchu pojazdów, tak by komunikacja miejska funkcjonowała sprawnie i bez zakłóceń. Jako mechanizm synchronizacji wykorzystaj semafory ogólne. Wykorzystaj notację *P(X,n)* i *V(X,n)* oznaczające odpowiednio opuszczenie i podniesienie semafora *X* o wartość *n*. Odpowiednie operacje należy wpisać w odpowiednie miejsca poniższej tabeli. W przypadku konieczności umieszczenia wielu operacji w jednym polu, ich kolejność ma znaczenie !

❶	❷	❸	❹
BE	BC	BA	BE
EH	CF	AD	EH
HG	FI	DG	HI
GD	IH	GH	IF
DA	HG	HI	FC
AB	GD	IF	CB
	DE	FE	
	EB	EB	



Rys. 2 Schemat linii komunikacyjnych

5. W pewnym starym angielskim mieście firma przewozowa postanowiła wprowadzić nową usługę dla turystów – przejazd przez stare miasto zabytkowymi autobusami. Turystom zaproponowano 4 tego trasy oznaczone odpowiednio literami A, B, C, D.
Z uwagi na wąskie uliczki w obrębie starego miasta, kierowcy autobusów są zmuszeni do maksymalnej uwagi, gdyż nie jest możliwe minięcie się dwóch autobusów jadących w przeciwnych kierunkach (za wyjątkiem skrzyżowań, które są nieco szersze). Ze względu na konstrukcję pojazdu i ograniczone pole manewru cofanie autobusów nie jest możliwe. Zatrzymanie autobusu jest dozwolone jedynie przed skrzyżowaniem.

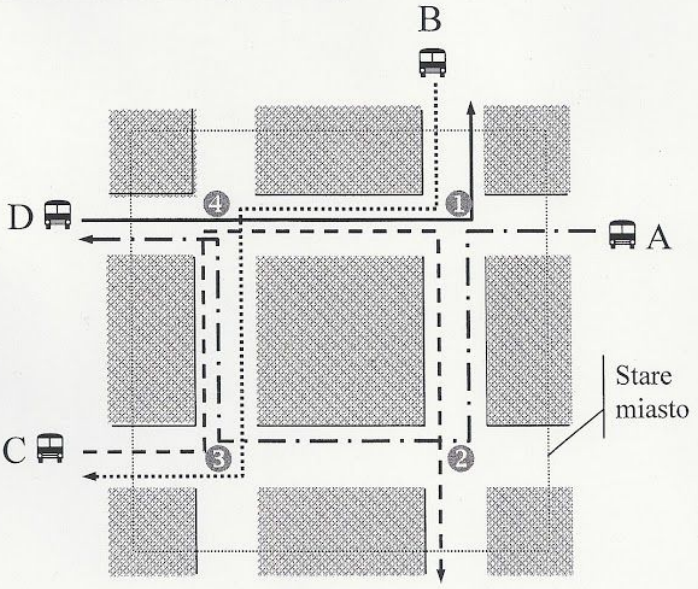
Napisz pseudokod dla kierowcy autobusu kursującego na każdej z tras. Powinien on zawierać elementy poniższych planów tras i operacje synchronizujące P i V na semaforach uogólnionych (operacje P(X,n) oraz V(X,n) oznaczają odpowiednio opuszczenie/podniesienie semafora X o n jednostek]

Trasa A:
wjazd na stare miasto,
przejazd przez ❶
przejazd przez ❷
przejazd przez ❸
przejazd przez ❹
wyjazd ze starego miasta

Trasa B:
wjazd na stare miasto,
przejazd przez ❶
przejazd przez ❹
przejazd przez ❸
wyjazd ze starego miasta

Trasa C:
wjazd na stare miasto,
przejazd przez ❸
przejazd przez ❹
przejazd przez ❶
przejazd przez ❷
wyjazd ze starego miasta

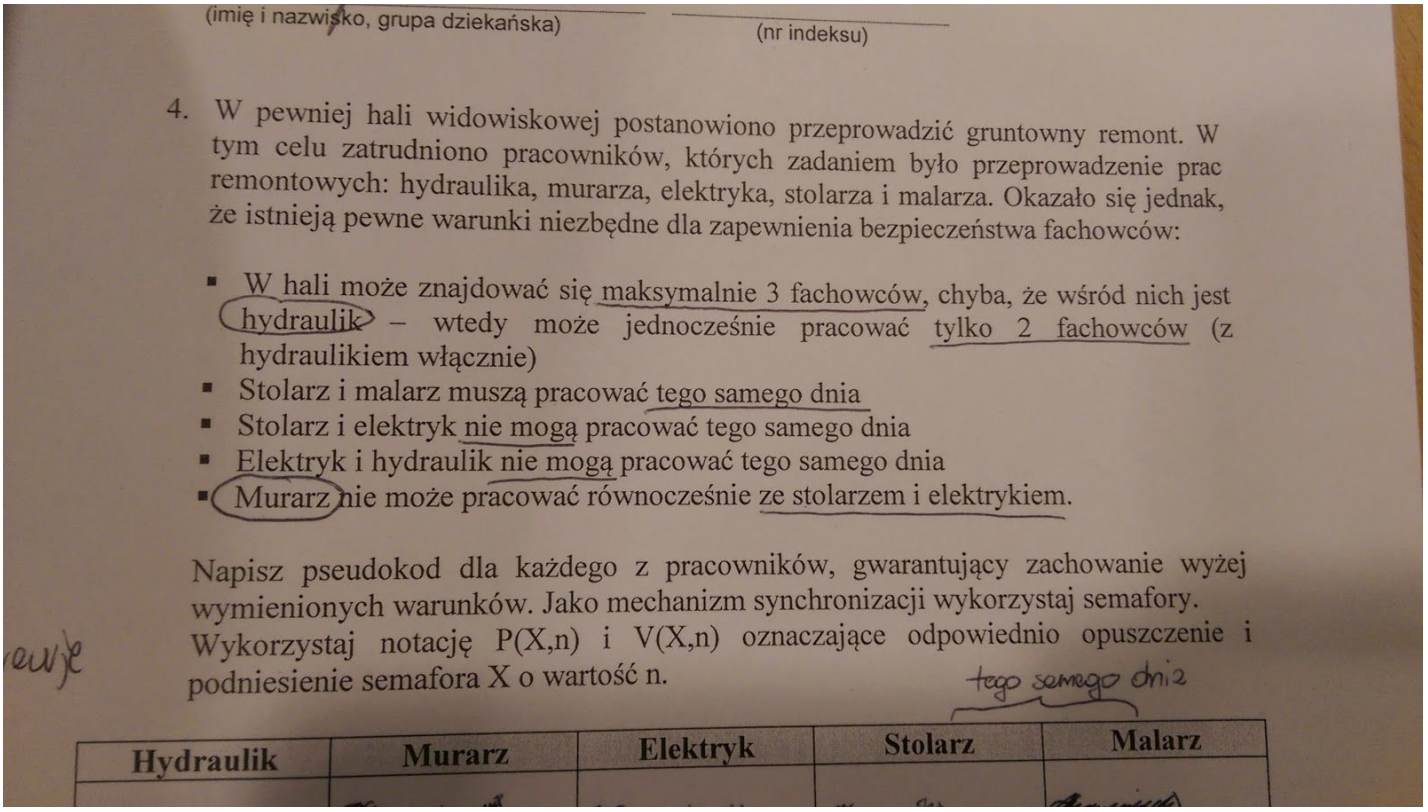
Trasa D:
wjazd na stare miasto,
przejazd przez ❹
przejazd przez ❶
wyjazd ze starego miasta



// wymyśliłem coś takiego, proszę o weryfikację bo mogę nie ogarniać (miałem 0 pkt z tego)
// Jak blokowałem skrzyżowania to miałem całe 0 punktów:)
// z tego co pamiętam kryterium też miała być minimalizacja liczby semaforów
// To raczej nie przejdzie
//Ja miałam rozwiązanie blokujące dane dwa kolidujące autobusy (w uj semaforów, chyba z 7) i dostałam 3p. Coś w stylu: w pkt 1 trzeba było blokować autobusy A i C, bo jechały w tę samą ulicę. Ale to też kiepskie

```
jedz - funkcja przebywania drogi w czasie

var
      S1, S2, S3, S4 : Semaphore;

procedure A
begin
      P(S1, 1);
      P(S2, 1);
            jedz;
      P(S3, 1);
      P(S2, 1); // przy skrecie w lewo
      V(S1, 1);
            jedz;
      P(S4, 1);
      V(S2, 2);
            jedz
      P(S4, 1); // przy skrecie w lewo
      V(S3, 1);
            jedz
            V(S4, 2);
end;

procedure B
begin
      P(S4, 2);
      P(S1, 1);
            jedz
      P(S3, 2);
      V(S1, 1);
            jedz
      V(S4, 2);
            jedz
      V(S3, 2);
end;

procedure C
```

```
begin
      P(S3, 1);
      P(S4, 1);
            jedz
      P(S1, 1);
      V(S3, 1);
            jedz
      P(S2, 1);
      V(S4, 1);
            jedz
      V(1, 1);
      V(2, 1);
end;

procedure D
begin
      P(S4, 1);
      P(S1, 1);
            jedz
      V(4, 1);
            jedz
      V(1, 1);
end;

begin
      Init_Semaphore(S1, 2);
      Init_Semaphore(S2, 2);
      Init_Semaphore(S3, 2);
      Init_Semaphore(S4, 2);
      parbegin
            A;
            B;
            C;
            D;
      parend;
end;
```



(imię i nazwisko, grupa dziekańska) (nr indeksu)

4. W pewniej hali widowiskowej postanowiono przeprowadzić gruntowny remont. W tym celu zatrudniono pracowników, których zadaniem było przeprowadzenie prac remontowych: hydraulika, murarza, elektryka, stolarza i malarza. Okazało się jednak, że istnieją pewne warunki niezbędne dla zapewnienia bezpieczeństwa fachowców:

- W hali może znajdować się maksymalnie 3 fachowców, chyba, że wśród nich jest hydraulik – wtedy może jednocześnie pracować tylko 2 fachowców (z hydraulikiem włącznie)
- Stolarz i malarz muszą pracować tego samego dnia
- Stolarz i elektryk nie mogą pracować tego samego dnia
- Elektryk i hydraulik nie mogą pracować tego samego dnia
- Murarz nie może pracować równocześnie ze stolarzem i elektrykiem.

Napisz pseudokod dla każdego z pracowników, gwarantujący zachowanie wyżej wymienionych warunków. Jako mechanizm synchronizacji wykorzystaj semafory. Wykorzystaj notację P(X,n) i V(X,n) oznaczające odpowiednio opuszczenie i podniesienie semafora X o wartość n.

tego samego dnia

Hydraulik	Murarz	Elektryk	Stolarz	Malarz

Czy murarz nie może pracować jednocześnie ze stolarzem i elektrykiem, czy może nie może pracować murarz z elektrykiem i murarz ze stolarzem? Poniższe rozwiązanie zakłąda pierwszą możliwość, stolarz i elektryk nie pracują razem więc nie trzeba rozważać ostatniego podpunktu

I(EH,1)
I(osoby,3)
I(SE,1)
I(St,0)
I(Ma,1)

hydraulik	murarz	elektryk	stolarz	malarz
-		P(EH, 1)		
	-		P(osoby, 2) (rezerwuje miejsce dla malarza)	
P(EH, 1)		-	P(SE, 1)	
		P(SE, 1)	-	V(St, 1) P(Ma, 1)
			P(St, 1) V(Ma, 1)	-
P(osoby, 2)	P(osoby, 1)	P(osoby, 1)		
pracuje	pracuje	pracuje	pracuje	pracuje
V(osoby,2)	V(osoby,1)	V(osoby, 1)	V(osoby, 2)	
V(EH,1)		V(SE,1)	V(SE,1)	
		V(EH,1)		

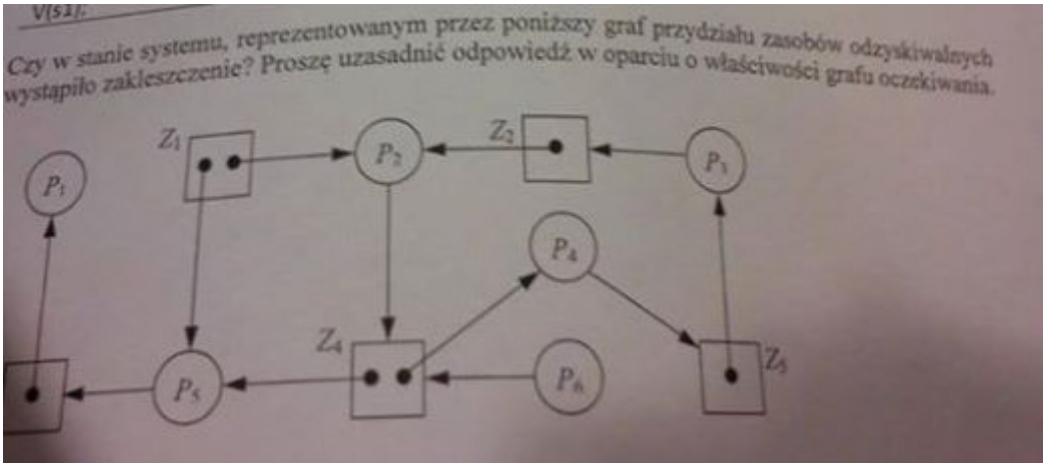
JEst Okey :P

1`

Tabelka wyzej starczy..
moje rozwiązanie zezwala na pracę robotnika tylko raz dziennie - gdy pracownik wyjdzie z hali danego dnia to już nie wejdzie

```
var
	osoby, SE, EH, St, Ma : SEMAPHORE;

procedure hydraulik
begin



	P(EH, 1);
	P(osoby, 2);
	pracuje
	V(osoby, 2);



	//V(EH, 1);
end;

// ok już rozumiem

procedure murarz
begin
	P(osoby, 1);
	pracuje
	V(osoby, 1);
end;

procedure elektryk
begin
	P(EH, 1);
	P(SE, 1);
	P(osoby, 1);
	pracuje
	V(osoby, 1);
	//V(SE, 1);
	//V(EH, 1);
end;

procedure stolarz


begin
	P(SE, 1);
	P(St, 1);
	V(Ma, 1);
	P(osoby, 2);
	pracuje
	V(osoby, 2);
	//V(SE, 1);

end;

procedure malarz
begin
	V(St, 1);
	P(Ma, 1);
	//P(osoby, 1);
	pracuje;
	//V(osoby, 1);

end;

procedure nowydzien
begin
	wait24h;
	Init_Semaphore(EH, 1);
	Init_Semaphore(SE, 1);
end;

begin
	Init_Semaphore(osoby, 3);
	Init_Semaphore(SE, 1);
	Init_Semaphore(EH, 1);
	Init_Semaphore(St, 0);
	Init_Semaphore(Ma, 0);
	parbegin
		hydraulik;
		murarz;
		elektryk;
		stolarz;
		malarz;
		nowydzien;
	parend;
end;
```

1. Dany jest system składający się z m jednostek zasobu tego samego typu, współdzielonych przez n procesów. Każdy z procesów może ubiegać się/zwalniać nie więcej niż jeden zasób w danym momencie czasu. Wykaż, że w systemie tym nie dojedzie do zakleszczenia, jeśli spełnione ~~się~~ są następujące dwa warunki:
   - Dla każdego z procesów maksymalna deklarowana liczba żądanych jednostek zasobu wynosi od 1 do m.
   - Suma maksymalnych deklarowanych jednostek zasobu żądanych przez procesy jest mniejsza niż $m + n$.



Nie, warunkiem koniecznym i wystarczającym w grafie z zasobami nie pojedyńczymi jest istnienie supła, a w tym grafie nie ma supła..

Istniejący cykl, który składa się z P_2, Z_4, P_4, Z_5, P_3, Z_2 posiada zasoby, które mogą zostać zwolnione (Z_4 o ile P_1 zwolni Z_3, a P_5 zwolni właśnie Z_4). Jeżeli procesy P_1 i P_5 nie zażądają kolejnych zasobów, to P_2 zarezerwuje Z_4, po wykonanej operacji Z_2 przejmie P_3, a jedną część Z_4 przejmie P_6. P_6 i P_3 się zakończą. P_3 zwolni Z_5 dla P_4, ten również się wykona.

Poznań, 25.01.2012

(imię i nazwisko, grupa dziekańska) (nr indeksu)

2. Udowodnij, że następujące rozwiązanie problemu wzajemnego wykluczania, zaproponowane przez Manna i Pnueliego, spełnia warunek bezpieczeństwa, przy założeniu, że operacja `if` jest atomowa.

integer $chce_p \leftarrow 0$, $chce_q \leftarrow 0$

p	q
loop forever p 1: sekcja lokalna p 2: **if** $chce_q = -1$ $chce_p \leftarrow -1$ **else** $chce_p \leftarrow 1$ p 3: **await** $chce_q \neq chce_p$ p 4: *sekcja krytyczna* p 5: $chce_p \leftarrow 0$	**loop forever** q 1: sekcja lokalna q 2: **if** $chce_p = -1$ $chce_q \leftarrow -1$ **else** $chce_q \leftarrow -1$ q 3: **await** $chce_p \neq -chce_q$ q 4: *sekcja krytyczna* q 5: $chce_q \leftarrow 0$

Instrukcja `await` jest niezależną od konkretnej implementacji notacją oznaczającą instrukcję, która oczekuje aż jej warunek będzie prawdziwy.
Znajdź scenariusz pokazujący, że algorytm jest niepoprawny, jeśli instrukcja `if` nie jest atomowa.

Zadanie z myjnią samochodową. 4 sterowniki (A - Pobierający opłatę, B - myjący koła, C - myjący nadwozie, D - suszący samochód). Najpierw pobrana opłata x (dostępne globalnie), po czym rozpoczyna się mycie kół (na raz myte jest jedno koło), po skończeniu mycia drugiego koła rozpoczyna się mycie nadwozia (składające się z x cykli), a po zakończeniu mycia wszystkich 4 kół i nadwozia - suszenie. Powierzchnia myjni ograniczona, więc mieści się tylko 1 samochód.

Poprawny kod problemu producent konsument z n-elementowym magazynem, oparty na semaforach. Napisać, co stanie się jak zmieniona zostanie (w procesie producenta) kolejność operacji: a) sprawdzenie czy istnieje miejsce w magazynie, dostęp do magazynu; na dostęp do magazynu, sprawdzenie czy istnieje miejsce w magazynie b) oddanie dostępu do magazynu, zwiększenie liczby elementów w magazynie; na zwiększenie liczby elementów w magazynie, oddanie dostępu do magazynu

a) producent zabiera dostep do magazynu, okazuje sie ze nie ma wolnego miejsca nigdy nie wlozy produktu do magazynu
b) opóźni tylko ?

// ktoś wie jak to zrobić ??????????????????
