Programowanie współbieżne - ćwiczenia 12
(monitory cz.1)

Zadanie 1 (Bar)

Napisz monitor BAR synchronizujący pracę barmana obsługującego klientów przy kolistym barze z N stołkami. Każdy klient realizuje następujący algorytm:

```
var
  i : integer;
begin
  repeat
    BAR.CZEKAJ_NA_STOŁEK_I_PIWO(i);
    <pije piwo na stołku i-tym>
    BAR.ZWALNIAM(i);
  until false;
end;
```

Barman nalewa piwo nowo przybyłym klientom w kolejności wyznaczonej przez stołki przez nich zajmowane (chodzi "w kółko"):

```
var
  i : integer;
begin
  repeat
    BAR.KTO_NASTĘPNY(i);
    <podejście do stołka i-tego i nalanie piwa>
    BAR.ZACZNIJ_PIĆ(i);
  until false;
end;
```

Rozwiązanie

```
monitor BAR;
var
  ilu_w: integer;       { ilu klientów jest w barze }
  WEJŚCIE: condition;   { tu klienci czekają na wejście do baru }

  stołek: array [0..N-1] of (wolny, nieobsłużony, obsłużony);
  { stan stołka: nieobsłużony - siedzący na nim klient czeka na piwo
                 obsłużony    - siedzący na nim klient pije piwo }

  czeka: array [0..N-1] of condition;  { tu klient czeka na nalanie piwa }
  ilu_czeka: integer;   { ilu klientów czeka na nalanie piwa }

  BARMAN: condition;    { tu czeka barman, jeżeli nie ma nic do roboty }
```

```
export procedure CZEKAJ_NA_STOŁEK_I_PIWO(var i: integer);
begin
  if ilu_w = N then wait(WEJŚCIE); { nie ma wolnych stołków }
  inc(ilu_w);
  inc(ilu_czeka);
  i := 0;
  while stołek[i] <> wolny do    { klient szuka wolnego stołka }
    i := (i + 1) mod N;
  stołek[i] := nieobsłużony;     { i-ty stołek był wolny, klient go zajmuje }
  if ilu_czeka = 1 then signal(BARMAN); { zwalnia barmana, jeżeli ten czeka }
  wait(czeka[i]);                { czeka na stołku i-tym na piwo }
end;

export procedure ZWALNIAM(i: integer);
begin
  dec(ilu_w);
  stołek[i] := wolny;
  signal(WEJŚCIE);
end;

export procedure KTO_NASTĘPNY(var i:integer);
begin
  if ilu_czeka = 0 then wait(BARMAN); { nie ma nieobsłużonych klientów }
  i := (i + 1) mod N; { barman szuka klienta począwszy od następnego miejsca
                        po tym, które ostatnio obsługiwał }
  while stołek[i] <> nieobsłużony do
    i := (i + 1) mod N;
end;

export procedure ZACZNIJ_PIĆ(i: integer);
begin
  stołek[i] := obsłużony;
  dec(ilu_czeka);
  signal(czeka[i]);
end;

begin
for i:=0 to N do
  stołek[i] := wolny;
end.
```

Uwagi

- Stan stołka musi być określony przez trzy wartości. Studenci zapewne zaproponują, żeby trzymać informację o tym, czy stołek jest wolny, czy zajęty, a rozróżniać stołki obsłużone od nieobsłużonych sprawdzając niepustość kolejek z nimi związanych. Nie jest to poprawne, ponieważ klient w procedurze `CZEKAJ_NA_STOŁEK_I_PIWO`

wykonuje `wait(czeka[i])` po `signal(BARMAN)`. Zgodnie z semantyką operacji `signal` po wykonaniu `signal(BARMAN)` zaczyna działać barman, który sprawdzając kolejkę `czeka[i]` stwierdziłby, że jest ona pusta, ponieważ klient nie wykonał jeszcze `wait(czeka[i])`! Ten sam problem wystąpiłby, jeżeli przyjęlibyśmy założenie, że nalanie piwa jest wewnętrzną czynnością w monitorze, czyli procedury `KTO_NASTĘPNY i ZACZNIJ_PIĆ` złączylibyśmy w jedną.

- Barman szukając nieobsłużonego klienta nie może zaczynać zawsze od tego samego miejsca, ponieważ prowadzi to do zagłodzenia.
- Dlaczego nie skorzystaliśmy z wygodniejszych struktur danych – takich jak lista zamiast tablicy? Ponieważ barman szukając klienta nie mógłby wziąć pierwszego z listy oczekujących – powinien obsługiwać wszystkich czekających klientów obok których przechodzi, niezależnie od tego w jakiej kolejności przyszli. Ten problem można rozwiązać porządkując listę według numerów stołków, przy których siedzą klienci, jednak wtedy pojawia się nowa trudność – od którego miejsca należy rozpocząć, żeby nie spowodować zagłodzenia? Jak widać rozwiązanie korzystające z tablicy jest dużo prostsze.

Zadanie 2 (Ładowanie cegieł)

Grupa N ($N > 0$) robotników ma załadować stertę cegieł na samochód. Każdy z nich musi wiedzieć od kogo ma odbierać cegły i komu podawać, czyli musi znać numery obu swoich sąsiadów (należy przyjąć, że sterta cegieł ma numer 0, a samochód $N+1$). Pracę można rozpocząć dopiero wtedy, gdy zgłoszą się wszyscy robotnicy. Po zakończeniu załadunku robotnicy przychodzą po wypłatę. Każdy podaje swoją stawkę za pracę, na podstawie której wylicza się stawkę średnią, którą otrzymują wszyscy. Każdy robotnik działa według schematu:

```
procedura Robotnik(i: integer);
const moja_stawka = ...;
var lewy, prawy, wypłata: integer:
begin
  repeat
    M.CHCĘ_PRACOWAĆ(i, lewy, prawy);
    podawaj_cegły(lewy, prawy);
    M.WYPŁATA(moja_stawka, wypłata);
    odpoczynek(wypłata);
  until false;
end;
```

Należy napisać monitor M. Nie wolno deklarować żadnych struktur rozmiaru N lub większego.

Rozwiązanie niepoprawne

Podajemy rozwiązanie niepoprawne, ponieważ studenci najczęściej próbują je rozwiązać w taki właśnie sposób:

```
monitor M;
var
  ilu, ostatni, stawka: integer;
  na_pozostałych: condition;

export procedure CHCĘ_PRACOWAĆ(i: integer, var lewy, prawy: integer);
begin
  inc(ilu);
  lewy := ostatni;
  ostatni := i;
  if ilu < N then wait(na_pozostałych)
             else ostatni:=N+1;
  prawy := ostatni;
  ostatni := i;
  dec(ilu);
  if not empty(na_pozostałych) then signal(na_pozostałych);
end;

export procedure WYPŁATA(moja_stawka: integer, var wypłata: integer);
begin
  inc(ilu);
  if ilu = 1 then ostatni := 0;
  stawka := stawka + moja_stawka;
  if ilu < N then wait(na_pozostałych)
  else stawka := stawka div N;
  wypłata : = stawka;
  signal(na_pozostałych);
end;

begin
  ilu := 0;
  ostatni := 0;
  stawka := 0;
end.
```

Każdy z procesów musi poznać swoich sąsiadów: lewego, czyli proces, który zgłosił się bezpośrednio przed nim (wystarczy zapamiętać numer ostatniego procesu, który wykonywał procedurę) i prawego, czyli proces, który zgłosił się **bezpośrednio po** nim – ten właśnie warunek nie jest spełniony.

Przeanalizujmy następujący scenariusz:

$N = 4$, kolejność zgłaszania się robotników: 2 4 3 1

```
   2          4          3          1

lewy:=0
ostatni:=2
wait
           lewy:=2
           ostatni:=4
           wait
                      lewy:=4
                      ostatni:=3
                      wait
                                 lewy:=3
                                 ostatni:=1
                                 ostatni:=N+1
                                 prawy:=N+1
                                 ostatni:=1;
                                 signal
prawy:=1 (?!)
ostatni:=2
signal
           prawy:=2 (?!)
           ostatni:=4;
           signal
                      prawy:=4 (?!)
                      ostatni:=3
                      signal (pusty)
```

Rozwiązanie poprawne

Aby poznać numer prawego sąsiada nie używając dodatkowych struktur danych należy odwrócić kolejność procesów wykorzystując do tego operację `signal`. Przypominamy, że `signal` wstrzymuje działanie procesu do momentu, gdy zwalniany przez niego proces opuści monitor. Procesy wstrzymane przy wykonaniu operacji `signal` zwalniane są w kolejności **odwrotnej** do tej w jakiej ją wykonywały.

```
export procedure CHCĘ_PRACOWAĆ(i: integer, var lewy, prawy: integer);
begin
  inc(ilu);
  lewy := ostatni;
  ostatni := i;
  if ilu < N then begin
```

```
                    wait(na_pozostałych);
                    signal(na_pozostałych);
                    prawy := ostatni;
                    ostatni := i;
          end else begin
                    prawy := N + 1;
                    ilu := 0;
                    signal(na_pozostałych);
          end;
end;
```

Przeanalizujmy to na naszym przykładzie:

```
   2          4          3           1

lewy:=0
ostatni:=2
wait
           lewy:=2
           ostatni:=4
           wait
                      lewy:=4
                      ostatni:=3
                      wait
                                 lewy:=3
                                 ostatni:=1
                                 prawy:=N+1
                                 signal
signal
           signal
                      signal (pusty)
                      prawy:=1
                      ostatni:=3
           prawy:=3
           ostatni:=4
prawy:=4
ostatni:=2
```

Należy zwrócić uwagę, że nie potrzeba osobnych kolejek do wstrzymywania procesów czekających na pracę i na wypłatę, ponieważ nie ma niebezpieczeństwa, że procesy z tych dwóch grup się przemieszają.