Programowanie współbieżne - ćwiczenia 11

(semafory/monitory – porównanie)

Zadanie 1 (Stolik dwuosobowy)

W systemie działa N par procesów. Procesy z pary są nierozróżnialne. Każdy proces cyklicznie wykonuje `własne_sprawy`, a potem spotyka się z drugim procesem z pary (`randka`) w kawiarni przy stoliku dwuosobowym. W kawiarni jest tylko jeden stolik. Procesy mogą zająć stolik tylko wtedy, gdy obydwa są gotowe do spotkania, ale odchodzą od niego pojedyńczo (w różnym czasie kończą wykonywanie procedury `randka`).

a) Zapisz treść procesu `P(j: 1..N)` używając do synchronizacji semaforów.
b) Zapisz treść procesu `P(j: 1..N)` i monitor `KAWIARNIA` synchronizujący ich działanie.

Rozwiązanie a

```
var
  NA_PARĘ : array [1..N] of binary semaphore := (0,...,0);
  para : array [1..N] of boolean := (False, ...,False);
  NA_STÓŁ : binary semaphore := 1;
  przy_stole : integer := 0;
  OCHRONA : binary semphore := 1;

process P(j:1..N)
begin
  repeat
    własne_sprawy;
    P(OCHRONA);
    if not para[j] then begin
                          para[j] := True;
                          V(OCHRONA);
                          P(NA_PARĘ[j]);
                        end
    else begin
           para[j] := False;
           V(OCHRONA);
           P(NA_STÓŁ);
           przy_stole := 2;
           V(NA_PARĘ[j]);
    end;
    randka;
    P(OCHRONA);
    dec(przy_stole);
    if przy_stole = 0 then begin
                             V(OCHRONA);
                             V(NA_STÓŁ);
```

```
                    end
    else V(OCHRONA);
  until false;
end;
```

Rozwiązanie b

```
monitor KELNER
var
  NA_PARĘ : array [1..N] of condition;
  NA_STÓŁ : condition;
  przy_stole : integer;

export procedure CHCĘ_STOLIK(j:integer)
begin
  if empty(NA_PARĘ[j]) then wait(NA_PARĘ[j])
  else begin
         if przy_stole > 0 then wait(NA_STÓŁ);
         przy_stole := 2;
         signal(NA_PARĘ[j]);
       end;
end;

export procedure ZWALNIAM
begin
  dec(przy_stole);
  if przy_stole = 0 then signal(NA_STÓŁ);
end;

begin
  przy_stole := 0;
end.

process P(j:1..N)
begin
  repeat
    własne_sprawy;
    KELNER.CHCĘ_STOLIK(j);
    randka;
    KELNER.ZWALNIAM;
  until false;
end
```

Należy zwrócić uwagę na następujące różnice:

- Nie trzeba synchronizować dostępu procesów do zmiennych monitora (brak jawnego odpowiednika semafora `OCHRONA`) dlatego, że procedur monitora nie może wykonywać wiele procesów jednocześnie
- Operacja `wait` jest zawsze blokująca w przeciwieństwie do operacji `P`, która zależy od wartości semafora. Dlatego w rozwiązaniu **a** jest: `P(NA_STÓŁ)`, a w rozwiązaniu **b**: `if przy_stole > 0 then wait(NA_STÓŁ);`
- Operacja `signal` na pustej kolejce nic nie robi w przeciwieństwie do operacji `V`, która wykonana na semaforze na którym żaden proces nie czeka powoduje jego podniesienie (zostawienie otwartej drogi)
- Można sprawdzać niepustość kolejki monitora, więc nie trzeba pamiętać jej stanu (brak tablicy `para`)

Zadanie 2 (Implementacja monitora przy pomocy semaforów)

Należy napisać odpowiedniki operacji monitorowych korzystając z semaforów:
1) Rozpoczęcie wykonywania procedury
2) Operacja `wait`
3) Operacja `signal`
4) Zakończenie wykonywania procedury

Rozwiązanie a

Wersja prostsza: zakładamy, że signal jest ostatnią instrukcją w procedurze monitora.

```
var
  MONITOR: binary semaphore := 1; {żeby zapewnić wzajemne wykluczanie
                                   przy wykonywaniu procedur monitora}
  KOLEJKA: binary semaphore := 0;
  ilu_czeka: integer := 0;

  1) P(MONITOR);

  2) inc(ilu_czeka);
     V(MONITOR);
     P(KOLEJKA);
     dec(ilu_czeka);

  3) if ilu_czeka > 0 then V(KOLEJKA)
     else V(MONITOR);

  4) V(MONITOR);
```

Dziedziczenie sekcji krytycznej jest konieczne, aby zachować semantykę operacji monitorowych – po wykonaniu `signal` wykonuje się zwolniony proces.

Rozwiązanie b

Wersja pełna: signal może się pojawić w dowolnym miejscu w procedurze.

```
var
  MONITOR: binary semaphore := 1; {żeby zapewnić wzajemne wykluczanie
                                   przy wykonywaniu procedur monitora}
  KOLEJKA: binary semaphore := 0;
  STOS: binary semaphore := 0;

  ilu_czeka_k: integer := 0;
  ilu_czeka_s: integer := 0;

  1) P(MONITOR);

  2) inc(ilu_czeka_k);
     if ilu_czeka_s > 0 then begin
                               dec(ilu_czeka_s);
                               V(STOS);
                             end
     else V(MONITOR);
     P(KOLEJKA);

  3) if ilu_czeka_k > 0 then begin
                               dec(ilu_czeka_k);
                               inc(ilu_czeka_s);
                               V(KOLEJKA);
                               P(STOS);
                             end;

  4) if ilu_czeka_s > 0 then begin
                               dec(ilu_czeka_s);
                               V(STOS);
                             end
     else V(MONITOR);
```

Procesy wykonujące `wait` i kończące procedurę monitora zwalniają procesy, które wykonały `signal` (oczekujące na semaforze `STOS`).
Należy zwrocić uwagę, że nie jest to dokładna implementacja, ponieważ za pomocą semaforów nie można zasymulować kolejki (procesy wykonujące `wait`) ani stosu (procesy wykonujące `signal`) – procesy oczekujące na semaforze są zwalniane w kolejności zapewniającej żywotność, ale nie możemy powiedzieć dokładnie w jakiej.