Programowanie współbieżne — scenariusz ćwiczeń nr 10
(semafory, cz. 2)

Zadanie 1.

W systemie działa `N` grup procesów. Każdy proces cyklicznie wykonuje procedurę `wlasne_sprawy`, a następnie procedurę `OBLICZ`, którą w tym samym czasie mogą wykonywać tylko procesy należące do tej samej grupy. Pierwszy proces z grupy może rozpocząć wykonywanie procedury `OBLICZ` tylko wówczas, gdy nikt inny jej nie wykonuje. Po zakończeniu wykonywania procedury `OBLICZ` procesy czekają aż wszystkie wykonujące ją procesy zakończą jej wykonywanie. Po zakończeniu procedury przez ostatni z wykonujących ją procesów działanie powinny rozpocząć oczekujące procesy z kolejnej grupy. Zapisz treść procesu `process P (gr : 1..N)` używając samaforów.

Uwagi wstępne.

W tym zadaniu synchronizacja działania procesów dotyczy dwóch kwestii: zapewnienia poprawnego wykonywania procedury `OBLICZ` oraz wspólnego zakończenia (jednego cyklu) przetwarzania.

Rozwiązanie tego zadania wymaga użycia następujących semaforów:
- semafora do ochrony zmiennych (semafor binarny),
- semafora, pod którym czekają pierwsze procesy z poszczególnych grup,
- tablicy semaforów (jeden semafor dla każdej grupy), pod którymi czekają kolejne procesy z poszczególnych grup,
- semafora do synchronizacji procesów, które zakończyły działanie.

Niezbędne informacje o stanie systemu obejmują:
- numer aktualnie działającej grupy,
- liczbę działających procesów,
- liczby procesów oczekujących na rozpoczęcie działania (w poszczególnych grupach),
- liczbę procesów oczekujących na zakończenie,
- dodatkowo liczbę czekających grup (aby w prosty sposób zbadać, czy ktokolwiek czeka).

Rozwiązanie

```
var Ochrona : binary semaphore := 1;  {ochrona wspolnych zmiennych}
    Pierwsi : binary semaphore := 0;  {tu czekaja pierwsi z grup}
    Reszta  : array[1..N] of binary semaphore := {all 0};  { tu pozostali}
    Koniec  : binary semaphore := 0;  {a tu procesy, ktore juz zakonczyly}

    kto    : integer := 0;            {numer dzialajacej grupy; 0 - brak}
    dziala : integer := 0;            {liczba dzialajacych procesow}
    ile    : array[1..N] of integer := {all 0};   {ile procesow czeka}
    ilegrup: integer := 0;            {ile grup czeka}
    ilekon : integer := 0;            {ilu czeka po zakonczeniu obliczen}
```

```
process P (gr : 1..N);
begin
  while true do begin
    wlasne_sprawy;
    P(Ochrona);
    if kto = 0 then  kto:= gr  {nikogo nie ma, moge dzialac}
    else
      if kto <> gr then begin  {dziala moja grupa -> wchodze, wpp. czekam}
        inc(ile[gr]);
        if ile[gr] = 1 then begin  {jestem pierwszy z grupy}
          inc (ilegrup);
          V(Ochrona);                  {zwalniam dostep do zmiennych}
          P(Pierwsi);                  { i grzecznie czekam}
          dec (ilegrup);
          kto:= gr;                    {a teraz dziala moja grupa}
        end
        else begin                 {kolejny proces z grupy}
          V(Ochrona);
          P(Reszta[gr]);               {czekam razem z kolegami z grupy}
        end;
        dec(ile[gr]);              {juz sie doczekalem}
      end;
    inc(dziala);                   {i zaraz zaczne dzialac}
    if ile[gr] > 0 then V(Reszta[gr])       {budze nastepnego}
    else V(Ochrona);                        {lub zwalniam dostep}
    OBLICZ;
    P(Ochrona);
    dec(dziala);
    if dziala > 0 then begin       {koledzy jeszcze pracuja}
        inc(ilekon);               { wiec musze poczekac}
        V(Ochrona);
        P(Koniec);
        dec(ilekon)                {dziedziczenie sekcji!}
    end;
    if ilekon>0 then V(Koniec)     {zwalniam czekajacego,}
    else                           { a ostatni proces z grupy}
      if ilegrup >0 then V(Pierwsi)  { budzi pierwszego z innej grupy}
      else begin
        kto:= 0;                   { lub zwalnia, bo nikt nie czeka}
        V(Ochrona)
      end
  end
end;
```

Komentarze do rozwiązania zadania 1.

W obydwu miejscach (czekanie na rozpoczęcie działania, czekanie po zakończeniu działania) procesy są budzone potokowo, czyli jeden proces budzi następny, przekazując mu sekcję krytyczną, a dopiero ostatni proces zwalnia sekcję krytyczną (tu: dostęp do zmiennych).

Pytanie: czy można (w tym rozwiązaniu lub w ogóle) zmienić sposób budzenia, tak aby jeden proces budził wszystkich? Czy dostaniemy poprawne i sprawne rozwiązanie, a jeśli tak, to jakie są wady i zalety obu podejść?

Przyjmujemy, że obowiązującą definicją semaforów jest definicja klasyczna. Analizę dla praktycznej definicji pozostawiamy czytelnikowi.

Rozważmy najpierw kwestię wspólnego kończenia pracy przez wszystkie procesy z grupy. Przyjmijmy, że ostatni proces budzi wszystkich, czyli wykonuje instrukcję postaci: `for i:= 1 to ilekon do V(Koniec);` i kończy działanie. Semafor `Koniec` (ogólny, a nie binarny) ma poprawną wartość i każdy czekający proces może kontynuować pracę. Co się jednak stanie, gdy jakiś proces 'zaśpi'? Następna grupa zaczyna działać i może skończyć zanim ten 'śpioch' opuści podniesiony dla niego semafor.
Konsekwencje: proces z nowej grupy nie czeka na kolegów, natomiast proces ze starej grupy ciągle czeka.

Konieczność dziedziczenia sekcji przy budzeniu pierwszego procesu z kolejnej grupy jest chyba oczywista. Gdyby nie bylo dziedziczenia, inny proces mógłby na podstawie stanu systemu (m.in. zmienna `kto`) stwierdzić, że może pracować i rozpocząć działanie. Chwilę później uprawniona grupa również rozpoczęłaby działanie. Nie możemy tego naprawić, gdyż nie wiemy który z oczekujących procesów zostanie obudzony.

Rozważmy teraz kwestię rozpoczęcia pracy przez grupę procesów z tej samej grupy. Pierwszy proces z grupy mógłby, po obudzeniu, budzić wszystkich swoich kolegów. Drobna wada (cecha?) takiego rozwiązania, to zrzucenie pewnej pracy na jeden z procesów, a nie równomierne obłożenie obowiązkami wszystkich (sprawność całości w zasadzie bez zmian). Odpowiednio operując licznikami (inaczej niż w przedstawionym rozwiązaniu!) można uniknąć sytuacji opisanej powyżej, czyli niepoprawnej pracy systemu.

Pozostaje jeszcze inna kwestia: proces może zostać wstrzymany zaraz po zwolnieniu dostępu do zmiennych (`V(Ochrona)`), a bezpośrednio przed wykonaniem operacji `P`. A wówczas mogłoby się zdarzyć, że pierwszy proces z grupy zostanie obudzony i podniesie semafor. Przy (nieco) niewłaściwej (a być może trudnej do zauważenia) modyfikacji liczników mogłoby się zdarzyć, że wszystkie procesy z grupy zakończyłyby działanie nie czekając na spóźnialskiego kolegę, który z kolei mógłby rozpocząć pracę w dowolnym terminie (bo semafor jest podniesiony).

Uwaga końcowa: przedstawione rozwiązanie jest poprawne, niezależnie od przyjętej definicji semaforów, jest jednakowo sprawne i bardziej sprawiedliwie dzieli obowiązki między wszystkie procesy.

Zadanie 2.

W systemie działa pewna liczba procesów zajmujących się przetwarzaniem danych. Każda praca jest wykonywana dokładnie przez K procesów (K > 1). Każdy proces (w nieskończonej pętli) zgłasza się do pracy, otrzymuje numer kolejny z przedziału od 1 do K, a następnie czeka na zgłoszenie się wszystkich K procesów. Ostatni (K-ty) proces inicjuje przetwarzanie (`function inicjuj() : DANE`)). Zainicjowane dane są następnie przetwarzane sekwencyjnie przez wszystkie procesy z tej grupy, począwszy od pierwszego procesu (tj. procesu, który przy zgłoszeniu otrzymał numer 1) aż do ostatniego (`procedure przetwarzaj(var dane : DANE; nr : 1..K)`). Po zakończeniu przetwarzania każdy proces ponownie zgłasza się do pracy. Zakończenie całej pracy następuje po zakończeniu jej przetwarzania przez wszystkie procesy z grupy.

Równocześnie może być wykonywanych co najwyżej MAX prac (opisanych wyżej; MAX $\geq$ 1). Przetwarzanie różnych prac może (i powinno) odbywać się równolegle, przy czym przetwarzanie danej pracy przez i-ty proces z danej grupy może rozpocząć się dopiero po zakończeniu przetwarzania poprzedniej pracy przez i-ty proces z poprzedniej grupy.

Zapisz przy użyciu semaforów treść procesów działających w tym systemie. Podaj początkowe wartości wszystkich semaforów.

Uwagi wstępne.

Zapewnienie wymaganej synchronizacji procesów oznacza konieczność użycia dwóch dwuwymiarowych tablic semaforów: semafory `Faza` służą do zapewnienia sekwencyjnego przetwarzania każdej pracy, natomiast semafory `Dalej` służą do właściwej synchronizacji procesów realizujących różne prace (nie za szybko).

Ograniczenie liczby jednocześnie wykonywanych prac może być zrealizowane (inaczej niż w przedstawionym rozwiązaniu) przy użyciu semafora ogólnego, zainicjowanego na liczbę prac dozwolonych do jednoczesnego wykonywania. Przedstawiona wersja rozwiązania jest nieco bardziej złożona, przedstawiam ją jednak dlatego, że wielu studentów wybierających tego typu rozwiązanie popełniało wiele podstawowych błędów synchronizacyjnych. Ponadto w tym drugim rozwiązaniu każdy proces musiałby najpierw opuścić semafor, a po stwierdzeniu, że nie jest ostatnim z grupy (tu akurat ostatni, a nie pierwszy proces to robi) musiałby podnosić niesłusznie opuszczony semafor. Alternatywne rozwiązanie jest w sumie nieco krótsze, ale mniej eleganckie i nieco bardziej złożone.

`Przykładowe rozwiązanie.`

```
const K = ...;  MAX = ...;
var Faza  : array[1..K, 1..MAX] of binary semaphore = {all 0};
    Dalej : array[1..K, 1..MAX] of binary semaphore = {K*1; others 0};
    Prace : binary semaphore = 0;      {tylko MAX prac, reszta czeka}
    Ochrona : binary semaphore = 1;    {standardowa ochrona zmiennych}

    dane : array[1..MAX] of DANE;  {przetwarzane dane}
    ilePrac : integer = 0;         {liczba prac w toku}
    ileCzeka : integer = 0;        {liczba procesow czekajacych na Prace}
    praca : integer = 1;         {indeks w dane dla nastepnej pracy}
    ileProcesow : integer = 0;  {liczba procesow do nastepnej pracy}
```

```
process Proces;
var nr : integer;       {numer kolejny (1..K)}
    nrPracy : integer;  {numer pracy (1..MAX)}
begin
  while true do begin
    P(Ochrona);
    if ilePrac = MAX then begin    {MAX prac w toku, czekamy}
      inc(ileCzeka);
      V(Ochrona);
      P(Prace);
      dec(ileCzeka)   {sekcja odziedziczona}
    end;
    inc(ileProcesow);
    nr:= ileProcesow;   {moj numer kolejny}
    nrPracy:= praca;    {numer pracy}
    if nr = K then begin          {grupa w komplecie}
      inc(ilePrac);
      praca:= praca mod MAX + 1;  {nastepna praca}
      ileProcesow:= 0;            {jeszcze nie ma do niej nikogo}
      dane[nrPracy]:= inicjuj();
      V(Ochrona);
      V(Faza[1, nrPracy]);        {pierwszy z mojej grupy do roboty}
    end
    else
      if ileCzeka > 0 then V(Prace)  else  V(Ochrona);

    P(Faza[nr, nrPracy]);      {poprzednik z grupy zakonczyl}
    P(Dalej[nr, nrPracy]);     {nie za szybko (poprzednia praca)}
    przetwarzaj( dane[nrPracy], nr );
                               {kolejne prace moga juz byc kontynuowane}
    V(Dalej[nr, nrPracy mod MAX + 1]);
    if nr < K then begin
      V(Faza[nr+1, nrPracy])   {nastepny z grupy do roboty}
    else begin              {zakonczenie pracy}
      P(Ochrona);
      dec(ilePrac);
      if ileCzeka > 0 then V(Prace)
      else V(Ochrona)
    end
  end
end;
```

Gdyby przyjąć (dospecyfikować), że rozpoczęcie pracy następuje już od zbierania procesów do jej wykonania, to wspomniane wcześniej alternatywne rozwiązanie niespełniałoby wymagań tego zadania. A wówczas przedstawione rozwiązanie byłoby (chyba) optymalne.