# Semafory, cz. 1

**Problem 1.** Implementacja różnych odmian semaforów.

(a) **Implementacja semafora dwustronnie ograniczonego.**

```
var S : semaphore := K;      {wartosc poczatkowa, 0 <= K <= N}
    T : semaphore := N - K;

procedure PD;   {operacja P na semaforze dwustronnie ograniczonym}
begin
  P(S);  V(T);
end;

procedure VD;   {operacja V na semaforze dwustronnie ograniczonym}
begin
  P(T);  V(S);
end;
```

**Uwaga.** Warto zauważyć, że zmiana kolejności wywołań w procedurze `VD` prowadzi do niepoprawnego rozwiązania (bardzo często podawanego przez studentów). Najlepiej to widać na przykładzie zastosowania powyższego schematu do rozwiązania problemu producentów i konsumentów z ograniczonym buforem.

**Zastosowanie — producent i konsument, bufor cykliczny.**

```
var bufor : array[0..N-1] of porcja;   {bufor cykliczny}
    Wolne  : semaphore := N;           {bufor inicjalnie pusty}
    Zajete : semaphore := 0;

process Producent;
var k : integer := 0;
    p : porcja;
begin
  while true do begin
    produkuj(p);
    P(Wolne);
    bufor[k]:= p;
    V(Zajete);
    k:= (k + 1) mod N
  end
end;

process Konsument;
var k : integer := 0;
    p : porcja;
begin
  while true do begin
    P(Zajete);
    p:= bufor[k];
    V(Wolne);
    k:= (k + 1) mod N;
    konsumuj(p)
  end
end;
```

(b) **Implementacja semafora uogólnionego.**

Należy zdefiniować dwie operacje, nazwijmy je `PG(n)` oraz `VG(n)`, oznaczające odpowiednio opuszczenie oraz podniesienie semafora uogólnionego.

W przedstawionym, prostym i zwięzłym, rozwiązaniu semafor (tzn. zmienna pomocnicza) przyjmuje czasowo wartości ujemne (sic!). Sposób działania procesów (wstrzymywanie procesów) jest jednak całkowicie prawidłowy (i o to przecież chodzi). Rozwiązanie, w którym zmienna pomocnicza nie przyjmuje wartości ujemnych jest znacznie bardziej złożone (por. Z. Weiss, T. Gruźlewski).

```
var mutex   : binary semaphore := 1;     {wylaczny dostep}
    ochrona : binary semaphore := 1;     {ochrona wspolnej zmiennej}
    Sczekaj : semaphore := 0;            {oczekiwanie}
    ile : integer;           {aktualna wartosc semafora}

procedure PG(n : integer);   {operacja P na semaforze uogolnionym}
begin
  P(mutex);          {co najwyzej jeden proces czeka na P}
  P(ochrona);
  ile:= ile - n;
  if ile < 0 then    {wstrzymanie}
  begin
    V(ochrona); P(Sczekaj);
  end
  else V(ochrona);
  V(mutex)
end;

procedure VG(n : integer);   {operacja V na semaforze uogolnionym}
begin
  P(ochrona);
  ile:= ile + n;
  if (ile >= 0) and (ile < n) then  {ktos czeka i juz sie doczekal}
    V(Sczekaj);
  V(ochrona);
end;
```

**Problem 2.** Czytelnicy i pisarze.

Rozwiązanie klasycznego problemu przy użyciu semaforów, bez żadnych priorytetów, czyli bez zagłodzenia. Treści obu rodzajów procesów są prawie identyczne — z dokładnością do nazw zmiennych plus spełnienie wymagania, że tylko jeden pisarz może przebywać w czytelni.

Schemat działania procesu danej grupy jest następujący: jeśli nie ma procesów z grupy przeciwnej, to wchodzę do czytelni (pisarze pojedynczo), w przeciwnym przypadku czekam (na odpowiednim semaforze). Po zakończeniu korzystania z czytelni

przez wszystkie procesy, które uzyskały takie prawo, ostatni wychodzący proces wpuszcza wszystkie oczekujące procesy z grupy przeciwnej (podnosząc semafor odpowiednią liczbę razy).

Do realizacji powyższego schematu należy użyć:

• czterech liczników: dwa dla czytelników oraz dwa dla pisarzy, oznaczające liczbę procesów z danej grupy, które zgłosiły chęć skorzystania z czytelni oraz tych, które uzyskały zgodę na wejście do czytelni (dla czytelników zgoda jest równoważna wejściu do czytelni, a pisarze muszą wchodzić pojedynczo);

• dwa semafory binarne: jeden (klasycznie) do ochrony wspólnych zmiennych (wyłączny dostęp), drugi do wpuszczania pisarzy do czytelni;

• dwa semafory ogólne, na których czekają odpowiednio czytelnicy oraz pisarze (oczekiwanie na opuszczenie czytelni przez procesy z przeciwnej grupy).

```
var jest_cz, akt_cz, czeka_p, jest_p : integer := (0, 0, 0, 0);
    Ochrona   : binary semaphore := 1;   {ochrona wspolnych zmiennych}
    Czytelnia : binary semaphore := 1;   {dostep dla jednego pisarza}
    Czytelnicy : semaphore := 0;         {czekajacy czytelnicy}
    Pisarze    : semaphore := 0;         {czekajacy pisarze}

process Czytelnik;
begin
  while true do begin
    wlasne_sprawy;
    P(Ochrona);
    jest_cz:= jest_cz + 1;            {licznik wszystkich czytelnikow}
    if jest_p = 0 then begin  {nie ma pisarza, mozna czytac}
      akt_cz:= akt_cz + 1;            {licznik czytelnikow w czytelni}
      V(Ochrona)
    end
    else begin                {sa pisarze ... }
      V(Ochrona);
      P(Czytelnicy)           { ... czytelnicy czekaja}
    end;
    CZYTAM;
    P(Ochrona);
    akt_cz:= akt_cz - 1;  jest_cz:= jest_cz - 1;
    if akt_cz = 0 then        {wychodzi ostatni czytelnik}
      while jest_p > akt_p do begin  {wpuszczamy wszystkich pisarzy}
        akt_p:= akt_p + 1;           {zaznaczajac, ze sa oni aktywni}
        V(Pisarze);
      end;
    V(Ochrona)
  end
end;
```

Treści obu rodzajów procesów są prawie identyczne — zamiast `CZYTAM` pisarz wykonuje `P(Pisarze); PISZE; V(Pisarze)`.

**Czytelnicy i pisarze — priorytet dla pisarzy (zagłodzenie czytelników).**

Rozwiązanie dające priorytet pisarzom można uzyskać modyfikując przedstawione powyżej rozwiązanie. Interesujące cechy poniższego rozwiązania:

• dziedziczenie sekcji krytycznej (tu: dostęp do zmiennych) — np. czytelnik budzony przez pisarza dziedziczy jego sekcję krytyczną (obowiązkowo!);

• po zakończeniu pracy przez pisarzy do czytelni wchodzą wszyscy czekający czytelnicy — każdy czytelnik wpuszcza następnego czekającego czytelnika (o ile taki jest, również z dziedziczeniem sekcji krytycznej).

Osoby zainteresowane innym rozwiązaniem tego problemu odsyłam do książki:
W. Iszkowski, M. Maniecki, "Programowanie współbieżne", WNT 1982.
Bardzo pouczające jest pełne zrozumienie przedstawionego tam rozwiązania (zwłaszcza sekwencji trzech operacji P).

```
var jest_cz, akt_cz, jest_p : integer := (0, 0, 0);
    Ochrona   : binary semaphore := 1;  {ochrona wspolnych zmiennych}
    Czytelnia : binary semaphore := 1;  {dostep dla pisarza/czytelnikow}
    Czytelnicy : binary semaphore := 0; {czekajacy czytelnicy}

process Pisarz;
begin
  while true do begin
   wlasne_sprawy;
   P(Ochrona);
   jest_p:= jest_p + 1;
   V(Ochrona);
   P(Czytelnia);  PISANIE;  V(Czytelnia);
   P(Ochrona);
   jest_p:= jest_p - 1;
   if (jest_p = 0) and (jest_cz > 0) then  {wchodza czytelnicy}
   begin
     P(Czytelnia);       {czytelnie zajmuja czytelnicy; troche nieladne}
     V(Czytelnicy)       {uwaga: dziedziczenie sekcji krytycznej}
   end
   else V(Ochrona)
  end
end;

process Czytelnik;
begin
  while true do begin
    wlasne_sprawy;
    P(Ochrona);
    jest_cz:= jest_cz + 1;          {licznik wszystkich czytelnikow}
    if jest_p = 0 then begin  {nie ma pisarza, mozna czytac}
      akt_cz:= akt_cz + 1;          {licznik czytelnikow w czytelni}
```

```
      if akt_cz = 1 then P(Czytelnia);  {pierwszy czytelnik musi zajac}
      V(Ochrona)
    end
    else begin                {sa pisarze ... }
      V(Ochrona);
      P(Czytelnicy)           { ... czytelnicy czekaja}
      akt_cz:= akt_cz + 1;
      if jest_cz > akt_cz then V(Czytelnicy)   {wpuszczam nastepnego}
      else V(Ochrona)                          {a ostatni zwalnia}
    end;
    CZYTAM;
    P(Ochrona);
    akt_cz:= akt_cz - 1;  jest_cz:= jest_cz - 1;
    if akt_cz = 0 then        {wychodzi ostatni czytelnik}
      V(Czytelnia);           { - moze wejsc pisarz}
    V(Ochrona)
  end
end;
```

Możliwe jest również zapisanie treści obu procesów w sposób jeszcze bardziej zbliżony do poprawnego rozwiązania (bez priorytetów) — ale mniej ciekawy :-). Pozostawiamy to zainteresowanym czytelnikom.