Programowanie współbieżne

Ćwiczenia 2 – semafory cz. 1

Zadanie 1: Producent i konsument z buforem cyklicznym

```
type porcja;
void produkuj(porcja &p);
void konsumuj(porcja p);

porcja bufor[N];            /* bufor cykliczny */
semaphore wolne = N;        /* bufor jest początkowo pusty */
semaphore zajete = 0;

  process Producent() {
    int k = 0;
    porcja p;

    while (true) {
      produkuj(p);
      P(wolne);
      bufor[k] = p;
      V(zajete);
      k = (k + 1) % N;
    }
  }

  process Konsument() {
    int k = 0;
    porcja p;

    while (true) {
      P(zajete);
      p = bufor[k];
      V(wolne);
      k = (k + 1) % N;
      konsumuj(p);
    }
  }
```

Zadanie 2: Producent i wielu konsumentów z buforem cyklicznym

```
type porcja;
void produkuj(porcja &p);
void konsumuj(porcja p);

porcja bufor[N];            /* bufor cykliczny */
semaphore wolne = N;        /* bufor jest początkowo pusty */
semaphore zajete = 0;
int gk = 0;
binary semaphore ochrona = 1;

  process Producent() {
    int k = 0;
    porcja p;

    while (true) {
      produkuj(p);
      P(wolne);
      bufor[k] = p;
      V(zajete);
      k = (k + 1) % N;
    }
  }

  process Konsument[K] {
    porcja p;
    while (true) {
      P(zajete);
      P(ochrona);
      p = bufor[gk];
      gk = (gk + 1) % N;
      V(ochrona);
      V(wolne);
      konsumuj(p);
    }
  }
```

Zadanie 3: Semafor uogólniony

Zdefiniuj dwie operacje, nazwijmy je PG(n) oraz VG(n), oznaczające odpowiednio opuszczenie oraz podniesienie semafora uogólnionego o n wartości. Zwróć uwagę na zachowanie własności żywotności.

Rozwiązanie bez dziedziczenia sekcji krytycznej

```
binary semaphore ochrona = 1;   /* ochrona wspólnych zmiennych */
binary semaphore reszta = 1;    /* semafor dla reszty czekających */
binary semaphore pierwszy = 0;  /* semafor dla pierwszego czekającego */
int ile;                        /* aktualna wartość semafora */

void PG(int n)   /* operacja P na semaforze uogólnionym */
{
  P(reszta);                    /* zapewnia, że co najwyżej jeden proces czeka na semaforze pierwszy */
  P(ochrona);
  ile = ile - n;
  if (ile < 0) {                /* wstrzymanie */
    V(ochrona);
    P(pierwszy);
  }
  else V(ochrona);
  V(reszta);
}

void VG(int n)   /* operacja V na semaforze uogólnionym */
{
  P(ochrona);
  ile = ile + n;
  if ((ile >= 0) && (ile < n))   /* ktoś czeka i już się doczekał */
    V(pierwszy);
  V(ochrona);
}
```

W przedstawionym rozwiązaniu zmienna reprezentująca wartość semafora przyjmuje czasowo wartości ujemne. Sposób działania procesów (wstrzymywanie procesów) jest jednak całkowicie prawidłowy.

Rozwiązanie z dziedziczeniem sekcji krytycznej

```
binary semaphore ochrona = 1;   /* ochrona wspólnych zmiennych */
binary semaphore reszta = 1;    /* semafor dla reszty czekających */
binary semaphore pierwszy = 0;  /* semafor dla pierwszego czekającego */
int ile;                        /* aktualna wartość semafora */
int naileczeka;                 /* na ile czeka pierwszy czekający */

void PG(int n)        /* operacja P na semaforze uogólnionym */
{
  P(reszta);
  P(ochrona);
  if (ile < n) {
    naileczeka = n;
    V(ochrona);
    P(pierwszy);      /* dziedziczenie sekcji krytycznej */
```

```
    naileczeka = 0;
  }
  ile = ile - n;
  V(ochrona);
  V(reszta);
}

void VG(int n)        /* operacja V na semaforze uogólnionym */
{
  P(ochrona);
  ile = ile + n;
  if ((naileczeka > 0) && (ile >= naileczeka))
    V(pierwszy);      /* przekazanie sekcji krytycznej */
  else
    V(ochrona);
}
```

Zadanie 4: Czytelnicy i pisarze

W systemie działają dwa typy procesów: czytelnicy i pisarze. Pisarze zapisują, a czytelnicy odczytują dane. Jeśli pisarz pisze, to żaden inny proces nie czyta ani nie pisze. Jednocześnie może (i powinno) czytać wielu czytelników.

Zanim napiszemy program, zastanówmy się, jaki jest ogólny schemat rozwiązywania tego typu zadań synchronizacyjnych. Mamy tutaj dwie grupy procesów, które muszą w jakiś sposób zsynchronizować korzystanie z *czytelni* – tak będziemy nazywać wykonanie operacji czytania przez czytelnika lub pisania przez pisarza. Nie znamy liczby czytelników ani pisarzy – może ich być dowolnie dużo. Podobnie w żaden sposób nie limitujemy liczby miejsc w czytelni.

Projektując protokoły wstępne odpowiadamy na pytanie „Kiedy proces powinien poczekać?". Z kolei pisząc protokoły końcowe podejmuje decyzję, którą grupę procesów czekających w protokołach wstępnych, obudzić. Spróbujmy zatem opisać protokoły wstępne i końcowe czytelnika i pisarza.

Protokół wstępny czytelnika: czytelnik czeka, jeśli w czytelni jest pisarz.

Protokół wstępny pisarza: pisarz czeka jeśli, w czytelni jest czytelnik lub pisarz.

Protokół końcowy czytelnika: jeśli jest ostatnim wychodzącym, to budzi pisarza o ile jakiś czeka.

Protokół końcowy pisarza: jeśli czekają czytelnicy, to inicjuje budzenie wszystkich czekających czytelników, w przeciwnym razie, jeśli czekają pisarze, to budzi jednego z nich.

Zanim zaczniemy implementować to rozwiązanie, odpowiedzmy na kilka pytań.

- Czy takie rozwiązanie jest bezpieczne?
- Czy nie prowadzi do zagłodzenia czytelników?
- Czy nie prowadzi do zagłodzenia pisarzy?

Przedstawione rozwiązanie może zagłodzić pisarzy. Wskaż scenariusz prowadzący do takiej sytuacji. Poprawne rozwiązanie polega na zatrzymaniu czytelnika także wtedy, gdy czeka pisarz. Protokół wstępny czytelnika będzie zatem następujący:

Protokół wstępny czytelnika: czytelnik czeka, jeśli w czytelni jest pisarz lub jeśli czeka pisarz.

Do realizacji powyższego schematu należy użyć:

- czterech liczników: dwóch dla czytelników oraz dwóch dla pisarzy, oznaczających liczby procesów z danej grupy, które zgłosiły chęć skorzystania z czytelni oraz tych, które już korzystają z czytelni;
- semafora binarnego do ochrony wspólnych zmiennych;
- dwóch semaforów, na których czekają odpowiednio czytelnicy oraz pisarze.

```
void czytam();
void piszę();

int iluczyta, ilupisze, czekaczyt, czekapis = (0, 0, 0, 0);
binary semaphore ochrona = 1;
semaphore Czytelnicy = 0;
semaphore Pisarze = 0;

process Czytelnik(int id) {
 while (true) {
   sekcja_lokalna;
   P(ochrona);
   if (ilupisze + czekapis > 0) {
     czekaczyt++;
     V(ochrona);
     P(Czytelnicy);   /* dziedziczenie sekcji krytycznej */
     czekaczyt--;
   };
   iluczyta++;
   if (czekaczyt > 0)
     V(Czytelnicy);
   else
     V(ochrona);
   czytam();
   P(ochrona);
   iluczyta--;
   if ((iluczyta == 0) && (czekapis > 0))
     V(Pisarze);
   else
     V(ochrona);
 }
}

process Pisarz(int id) {
 while (true) {
   sekcja_lokalna;
   P(ochrona);
   if (ilupisze + iluczyta > 0) {
     czekapis++;
     V(ochrona);
     P(Pisarze);   /* dziedziczenie sekcji krytycznej */
     czekapis--;
   };
   ilupisze++;
   V(ochrona);
   piszę();
```

```
    P(ochrona);
    ilupisze−−;
    if (czekaczyt > 0)
      V(Czytelnicy);
    else if (czekapis > 0)
      V(Pisarze);
    else
      V(ochrona);
  }
}
```

Zadanie dodatkowe

Zapisz rozwiązanie problemu czytelników i pisarzy bez dziedziczenia sekcji krytycznej.