Programowanie współbieżne

Ćwiczenia 3 – semafory cz. 2

Zadanie 1: Stolik dwuosobowy

W systemie działa N par procesów. Procesy z pary są nierozróżnialne. Każdy proces cyklicznie wykonuje *własnesprawy*, a potem spotyka się z drugim procesem z pary w kawiarni przy stoliku dwuosobowym (funkcja *randka*()). W kawiarni jest tylko jeden stolik. Procesy zajmują stolik tylko wtedy, gdy obydwa są gotowe do spotkania, natomiast mogą odchodzić od niego pojedynczo, czyli w różnym czasie kończyć wykonywanie funkcji *randka*().

Rozwiązanie

W rozwiązaniu użyjemy następujących semaforów:

- semafora do ochrony zmiennych,
- semafora, na którym czekają procesy, które przyszły jako pierwsze z pary,
- semafora, na którym drugie procesy z pary czekają na stolik.

Potrzebna będzie również informacja:

- dla każdej pary o tym, czy pierwszy proces z pary już czeka,
- liczbie procesów siedzących przy stoliku.

```
void randka();

binary semaphore ochrona = 1;
binary semaphore naparę[N] = (0,...,0);
binary semaphore nastół = 1;
bool para[N] = (false ,..., false );
int  przystole  = 0;


process P(int j) {            /* j to numer pary */
  while (true) {
    własnesprawy;
    P(ochrona);
    if (para[j] == false) { /* pierwszy z pary */
      para[j] = true;
      V(ochrona);
      P(naparę[j]);           /* dziedziczenie sekcji krytycznej */
      przystole++;
      V(ochrona);
    }
    else {                    /* drugi z pary */
      para[j] = false;
      V(ochrona);
      P(nastół);
      P(ochrona);
      przystole++;
      V(naparę[j]);           /* przekazanie sekcji krytycznej */
    }
```

```
    randka();

    P(ochrona);
    przystole−−;
    if (przystole == 0) V(nastół);
    V(ochrona);
  }
}
```

Zadanie 2: Synchronizacja grupowa

W systemie działa N grup procesów. Każdy proces cyklicznie wykonuje procedurę *własnesprawy* (sekcja lokalna), a następnie funkcję *oblicz*(), którą w tym samym czasie mogą wykonywać tylko procesy należące do tej samej grupy. Pierwszy proces z grupy może rozpocząć wykonywanie funkcji *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.

Rozwiązanie

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,
- semafora, na którym czekają pierwsze procesy z poszczególnych grup,
- tablicy semaforów (jeden semafor dla każdej grupy), na których 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.

void oblicz ();

```
binary semaphore ochrona = 1;          /* ochrona wspólnych zmiennych */
binary semaphore pierwsi = 0;          /* tu czekają pierwsi z grup */
binary semaphore reszta[N] = (0,...,0); /* tu pozostali */
binary semaphore koniec = 0;           /* a tu procesy, które już zakonczyły */

int kto = 0;                           /* numer działającej grupy; 0 oznacza, że nikt nie działa */
int iledziała = 0;                     /* liczba działających procesów */
int ileczeka[N] = (0,...,0);           /* ile procesów czeka */
int ilekończy = 0;                     /* ile czeka po zakończeniu obliczeń */
int ilegrup = 0;                       /* ile grup czeka */

process P (int gr) {
  while true do {
    własnesprawy;
    P(ochrona);
    if (kto == 0) kto = gr;    /* nikogo nie ma, moge działać */
    else {
      if (kto<>gr) {           /* jeśli nie działa moja grupa, to czekam */
        ileczeka[gr]++;
        if (ileczeka[gr] == 1) {      /* jestem pierwszy z grupy */
          ilegrup++;
          V(ochrona);
          P(pierwsi);                 /* czekam jako reprezentant grupy */
          ilegrup--;                  /* dziedziczę sekcję krytyczną */
          kto = gr;                   /* teraz działa moja grupa */
        }
        else {                        /* jestem kolejnym procesem z grupy */
          V(ochrona);
          P(reszta[gr]);              /* czekam razem z kolegami z grupy */
        }                             /* dziedziczę sekcję krytyczną */
        ileczeka[gr]--;
    }
    iledziała ++;                     /* zaraz zacznę działać */
    if (ileczeka[gr] > 0) V(reszta[gr]);  /* budzę nastepnego jeśli jest */
    else V(ochrona);                  /* wpp zwalniam dostęp */

    oblicz ();

    P(ochrona);
    iledziała --;
    if (iledziała > 0) {              /* koledzy jeszcze pracują */
        ilekończy++;                  /* więc muszę poczekać */
        V(ochrona);
        P(koniec);                    /* dziedziczenie sekcji krytycznej */
        ilekończy--;
    }
    if (ilekończy>0) V(koniec);       /* zwalniam czekającego, */
    else                              /* ostatni proces z grupy */
      if (ilegrup>0) V(pierwsi);      /* budzi pierwszego z kolejnej grupy */
      else { kto = 0;                 /* zwalnia dostęp, jeśli nikt nie czeka */
             V(ochrona);
           }
  }
}
```

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 (dostęp do wspólnych zmiennych).

Czy można zmienić sposób budzenia, tak aby jeden proces budził wszystkich? Czy dostaniemy poprawne rozwiązanie, a jeśli tak, to jakie są wady i zalety obu podejść?

Rozważmy najpierw kwestię wspólnego kończenia pracy przez wszystkie procesy z grupy. Przyjmijmy, że ostatni proces budzi wszystkich, czyli *ilekończy* razy wykonuje *V(koniec)* i kończy działanie. Semafor *koniec*, który musiałby w takim przypadku być ogólny, a nie binarny, ma poprawną wartość i każdy czekający proces może kontynuować pracę. Co się jednak stanie, gdy jeden obudzonych procesów „zaśpi"? Następna grupa zaczyna działać i może zakończyć obliczanie zanim ten „śpioch" opuści podniesiony dla niego semafor. W konsekwencji proces z nowej grupy nie będzie czekał na kolegów, natomiast „śpioch" ze starej grupy będzie wciąż czekał.

Konieczność dziedziczenia sekcji przy budzeniu pierwszego procesu z kolejnej grupy jest bardziej oczywista. Gdyby nie było 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.

Odpowiednio operując licznikami (inaczej niż w przedstawionym rozwiązaniu) można skonstruować poprawne rozwiązanie bez dziedziczenia sekcji krytycznej. Będzie ono jednak trochę bardziej złożone.