Programowanie współbieżne

Ćwiczenia 4 – semafory cz. 3

Zadanie 1: Taśma produkcyjna

Zadanie pochodzi z książki Weissa i Guźlewskiego „Programowanie współbieżne i rozproszone".

Taśma produkcyjna (bufor cykliczny) może pomieścić M elementów. Wokół taśmy chodzi N robotników. Zadaniem pierszego z nich jest umieszczenie nieobrobionego elementu na taśmie. Następny robotnik realizuje pierwszą fazę obróbki, kolejny drugą fazę itd. Ostatni zdejmuje gotowe elementy z taśmy robiąc miejsce pierwszemu, który może na zwolnione miejsce włożyć kolejny element. Każda faza obróbki może się rozpocząć tylko po zakończeniu porzedniej.

Rozwiązanie

```
void przetwarzaj(porcja element, int faza);

semaphore S[N] = (M,0,...0);
porcja dane[M];

process Robotnik(int i) {
int pozycja;
  while (true) {
    sekcja_lokalna;
    P(S[i]);
    przetwarzaj(dane[pozycja], i);
    V(S[(i+1)%N]);
    pozycja = (pozycja + 1)%N;
  }
}
```

Zadanie 2: Przetwarzanie potokowe

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 0 do K-1, a następnie czeka na zgłoszenie się wszystkich K procesów. Ostatni proces (o numerze *K-1*) inicjuje przetwarzanie (*porcja inicjuj*()). Zainicjowane dane są następnie przetwarzane sekwencyjnie przez wszystkie procesy z tej grupy, począwszy od pierwszego procesu (tj. procesu, który przy zgłoszeniu otrzymał numer 0) aż do ostatniego (*void przetwarzaj*(*porcja dane, int nr*)). 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 > 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.

Rozwiązanie

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).

```
binary semaphore Faza[K,MAX] = (0,...,0);
binary semaphore Dalej[K,MAX] = (K*1,0,...,0);
binary semaphore Praca = 0;      /* jednocześnie tylko MAX prac, reszta czeka */
binary semaphore Ochrona = 1;    /* ochrona zmiennych */

porcja dane[MAX];            /* przetwarzane dane */
int ilePrac = 0;             /* liczba prac w toku */
int ileCzeka = 0;            /* liczba procesow czekających na semaforze Praca */
int praca = 0;               /* indeks w dane dla następnej pracy */
int ileProcesów = 0;         /* liczba procesów do następnej pracy */

process Proces() {
int nr;                      /* numer kolejny od 0 do K-1 */
int nrPracy;                 /* numer pracy od 0 do MAX */

while (true) {
      P(Ochrona);
      if (ilePrac == MAX) { /* MAX prac w toku, czekamy */
        ileCzeka++;
        V(Ochrona);
        P(Praca);
        ileCzeka--;          /* sekcja odziedziczona */
      };
      nr = ileProcesów++; /* mój numer kolejny */
      nrPracy = praca;       /* numer pracy */
      if (nr == K) {                 /* grupa w komplecie */
        ilePrac++;
        praca = (praca+1) % MAX; /* następna praca */
        ileProcesów = 0;             /* jeszcze nie ma do niej nikogo */
        dane[nrPracy] = inicjuj ();
        V(Ochrona);
        V(Faza[1, nrPracy]);         /* pierwszy z mojej grupy może pracować */
      }
      else
        if (ileCzeka > 0) V(Praca) else V(Ochrona);

      P(Faza[nr, nrPracy]);      /* czekam aż poprzedni z grupy zakończy */
      P(Dalej[nr, nrPracy]);     /* czekam aż poprzednia praca na tym etapie się zakończy */

      przetwarzaj(dane[nrPracy], nr);

      V(Dalej[nr, (nrPracy+1)%MAX]); /* kolejne prace mogą już być kontynuowane */
      if (nr < K-1) {
        V(Faza[nr+1, nrPracy])  /* następny z grupy może pracować */
      else {                        /* zakończenie pracy */
        P(Ochrona);
        ilePrac--;
        if (ileCzeka > 0) V(Prace) else V(Ochrona);
    }
  }
}
```