Programowanie współbieżne

Ćwiczenia 7 – powtórzenie przed kolokwium

Zadanie 1: Biuro

W pewnym biurze grupa urzędników obsługuje grupę klientów. Każdy urzędnik ma unikatowy identyfikator. Algorytmy urzędników i klientów są następujące:

```
process Urzędnik (int ranga, int id) {

  while (true) {
    Biuro.ChcęPracować(ranga, id);
    // rozmawiam z klientem
    Biuro.Skończyłem;
    // odpoczywam
   }
}
```

```
process Klient () {
  int u1, u2;
  while (true) {
    Biuro.ChcęZałatwićSprawę(&u1, &u2);
    // rozmawiam z urzędnikami u1 i u2
    Biuro.Załatwiłem;
    // własne sprawy
  }
}
```

- W biurze pracuje co najmniej jeden urzędnik wyższej rangi lub co najmniej dwóch urzędników niższej rangi.
- Na wyposażenie biura składa się K (K $>$ 2) krzeseł, z których korzystają urzędnicy i klienci podczas rozmowy.
- Do rozmowy dochodzi, jeżeli jest zainteresowany klient i chętny do pracy urzędnik wyższej rangi lub dwóch urzędników niższej rangi oraz wystarczająca liczba krzeseł (każda osoba zajmuje jedno krzesło).
- Urzędnicy w ramach każdej z grup pracują według kolejności zgłaszania się do pracy.
- Klienci muszą dowiedzieć się z jakimi urzędnikami mają rozmawiać (muszą poznać ich identyfikatory).
- Po skończonej rozmowie urzędnicy i klienci mogą opuszczać krzesła pojedynczo.
- Biuro powinno pracować w ten sposób, aby obsłużyć jak największą liczbę klientów i jest to priorytetem w tym zadaniu.

Napisz monitor Biuro.

Rozwiązanie

```
monitor BIURO{

  int ileKrzeseł = K;
  int ilu2 = 0;                          /* ilu jest gotowych urzędników II */
  condition urzędnicy1, urzędnicy2;      /* czekają urzędnicy I i II */
  condition klienci;                     /* czekają klienci */
  int id1, id2 = (0,0);                  /* identyfikatory urzędników */

  void ChcęPracować(int ranga, int id) {
    if (ranga == 1) {
      /* czeka, jeżeli nie ma klientów lub wystarczającej liczby krzeseł */
      if (empty(k) || ileKrzeseł < 2) wait(urzędnik1);
```

```
      ileKrzeseł = ileKrzeseł − 2;
      id1 = id;            /* zapisuje swój identyfikator */
      id2 = 0;             /* informuje, że nie będzie drugiego urzędnika */
      signal(klient);      /* budzi klienta */
    } else {
      /* czeka, jeżeli nie ma klientów lub wystarczającej liczby
         krzeseł lub oczekującego urzędnika II */
      if (empty(klient) || ileKrzeseł < 3 || empty(urzędnicy2)) {
        ilu2++;
        wait(urzędnicy2);
        ilu2−−;
      }
      /* jeżeli jest pierwszym urzędnikiem II, to zapisuje
         swój identyfikator i budzi drugiego urzędnika */
      if (id1 == 0) {
        ileKrzeseł = ileKrzeseł − 3;
        id1 = id;
        signal(urzędnicy2);
      }
      /* jeżeli jest drugim urzędnikiem II, to zapisuje
         swój identyfikator i budzi klienta */
      else {
        id2 = id;
        signal(klienci);
      }
    }
  }

  void ChcęZałatwićSprawę(int &u1, &u2) {

    if ((empty(urzędnicy1) || ileKrzeseł < 2) /* jeżeli nie ma urzędnika I lub 2 krzeseł */
      && (ilu2 < 2 || ileKrzeseł < 3))        /* i nie ma dwóch urzędników II lub 3 krzeseł */
      wait(klienci);                          /* to klient czeka */
    else
      if (not empty(urzędnicy1)) signal(urzędnicy1); /* budzi piurzędnika I */
      else signal(urzędnicy2);                       /* budzi pierwszego urzędnika II */
    u1 = id1;                                        /* odczytuje identyfikatory */
    u2 = id2;
    id1 = 0;
  }

  void Skończyłem/Załatwiłem() {
    ileKrzeseł++;
    /* jeżeli może dojść do rozmowy, to zwalnia odpowiedniego urzędnika */
    if (not empty(k))
      if (not empty(urzędnicy1) && ileKrzeseł >= 2) signal(urzędnicy1);
      else if (ilu2 >= 2 && ileKrzeseł >= 3) signal(urzędnicy2);
  }
}
```

Uwagi

Schemat rozwiązania jest następujący: kiedy nie może dojść do rozmowy (z powodu braku jednej ze stron lub miejsc do siedzenia), wtedy proces czeka w odpowiedniej kolejce. Jeżeli pojawi się brakujący urzędnik pierwszej rangi, to zwalnia klienta. Jeżeli pojawi się brakujący urzędnik drugiej

rangi, to zwalnia drugiego urzędnika, który zwalnia klienta. Jeżeli pojawi się brakujący klient, to zwalnia urzędnika i jeżeli jest to urzędnik drugiej rangi, to zwalnia swojego kolegę, który zwolniłby klienta, ale tego nie robi, bo kolejka klientów jest pusta (gdyby nie była, to nie brakowało by klienta – patrz: początek zdania). Jeżeli pojawiają się brakujące krzesła, to zwalniani są odpowiedni urzędnicy, którzy zwalniają klienta.

Warto zwrócić uwagę na przekazywanie identyfikatorów klientowi. Jeżeli to klient zwalnia urzędników, to identyfikatory zostają przekazane w momencie, kiedy klient wykonuje signal.

Priorytetowym wymaganiem w tym zadaniu jest obsłużenie jak największej ilości klientów. Aby je spełnić należy w pierwszej kolejności wybierać urzędników pierwszej rangi, ponieważ zajmują oni mniej krzeseł. Oczywiście może dojść do zagłodzenia drugiej grupy urzędników, ale jest to zagłodzenie wynikające z treści zadania.

Zadanie 2: Mikst

Klub tenisowy organizuje rozgrywki miksta na N kortach. Do rozegrania meczu miksta potrzebne są dwa procesy typu 0, dwa procesy typu 1 oraz wolny kort. Napisz kod procesu *Gracz*(*int typ*), który cyklicznie wkonuje własne sprawy, czeka na partnerów do gry oraz wolny kort i rozgrywa mecz, czyli wywołuje procedurę *mecz*(*int nrKortu*). Do synchronizacji użyj semaforów.

Rozwiązanie

```
binary semaphore Ochrona = 1;
binary semaphore Gotowi = (0, 0);
int ileCzeka[2] = (0,0);
int ileWolnychBoisk = 0;
int ileGra[N] = (0,...,0);
int aktualneBoisko;

process Gracz(int typ){
  int boisko;

  while (true) {
    własne_sprawy;
    P(Ochrona);
    if (ileCzeka[typ] < 1 || ileCzeka[1−typ] < 2 || ileWolnychBoisk == 0){
      ileCzeka[typ]++;
      V(Ochrona);
      P(Gotowi[typ]);
      ileCzeka[typ]−−;
      boisko = aktualneBoisko;
      ileGra[boisko]++;
      if (ileGra[boisko]%2 == 1) V(Gotowi[typ]);
      else
        if (ileGra[boisko] == 2) V(Gotowi[1−typ]);
        else V(Ochrona);
    } else {
      boisko = 0;
      while (ileGra[boisko] <> 0) boisko++;
      aktualneBoisko = boisko;
      ileGra[boisko]++;
      ileWolnychBoisk−−;
      V(Gotowi[typ]);
    }
```

```
    mecz(boisko);

    P(Ochrona);
    ileGra[boisko]--;
    if (ileGra[boisko] == 0) {
      if (ileCzeka[typ] >= 2 && ileCzeka[1-typ] >= 2) {
        aktualneBoisko = boisko;
        V(Gotowi[typ]);
      } else {
        ileWolnychBoisk++;
        V(Ochrona);
      }
    }
    else V(Ochrona);
  }
}
```