Procesų sąveika

Procesų sąveikos tipai

- procesai nepriklausomi,
- procesai varžosi tarpusavyje,
- procesai bendradarbiauja tarpusavyje.

Procesai nepriklausomi

- LP kalboje – tik priemonės procesams kurti, vykdyti ir baigti.
- Priemonės procesų sąveikai valdyti – nereikalingos.
- Variantas retai pasitaikantis.

Procesai varžosi tarpusavyje

- LP kalboje – priemonės procesams kurti, vykdyti ir baigti.
- LP kalboje – priemonės procesų prioritetams nustatyti.
- Variantas, būdingas sistemoms, kuriose procesai sprendžia nepriklausomus uždavinius.

Procesai bendradarbiauja tarpusavyje

- LP kalboje – priemonės procesams kurti, vykdyti ir baigti.
- LP kalboje – priemonės procesų sąveikai valdyti.
- Variantas, būdingas programoms, kuriose procesai sprendžia bendrą uždavinį.

Reikalingos programavimo kalbos priemonės

- kurti ir vykdyti lygiagrečiuosius procesus,
- keistis informacija tarp procesų,
- sinchronizuoti procesų darbą.

Lygiagrečiųjų procesų kūrimas ir vykdymas

1. Kiekvieno proceso programos kodas – kitoks.
2. Visų procesų programų kodai sutampa.

Informacijos perdavimas tarp lygiagrečiųjų procesų

1. Procesai keičiasi informacija per bendrus kintamuosius.
2. Procesai keičiasi informacija siųsdami pranešimus (žinutes).

Lygiagrečiųjų procesų sinchronizavimas

1. Procesai laukia, kartodami kokius nors veiksmus.
2. Procesai laukia, neatlikdami jokių veiksmų (yra blokuojami).

Ar nuoseklaus programavimo kalbos priemonėmis galima:

- kurti ir vykdyti lygiagrečiuosius procesus?
- keistis informacija tarp procesų?
- sinchronizuoti procesų darbą?

Bendrųjų kintamųjų naudojimo problemos

- Kokia bus bendrojo kintamojo reikšmė, jei du procesai vienu metu keičia jo reikšmę?
- Vienas procesas rašo į kintamąjį, kitas naudoja (skaito). Kokią reikšmę perskaitys?

Bendras kintamasis-1. Programa

```
class Gija extends Thread {
  void run() {
    for (int i=0; i<DekSodas.n; i++)
      { int k=DekSodas.c; k++; DekSodas.c=k; }
  }
}
class DekSodas {
  int c = 0; int n = 10;
  void main(String[] args) {
    Gija g1 = new Gija();
    Gija g2 = new Gija();
    g1.start(); g2.start();
    System.out.println(c);
  }
}
```

Bendras kintamasis-1. Rezultatas

1. 15?
2. 20?
3. 10?

Kokia turi būti tikroji reikšmė?

Bendras kintamasis-2. Programa

```
// thread                          | // thread
class Siuntėjas {                  | class Gavėjas {
  int m;                           |   int d;
  void run() { ...                 |   void run() { ...
    m = 10;//Formuoti(m);          |     d = BK2.v; ...
    BK2.v = m; ...                 |     Naudoti(d); ...
} }                                | } }
class BK2 {
  int v = 0;
  void main() {
    Siuntėjas s = new Siuntėjas();
    Gavėjas g = new Gavėjas();
    s.start(); g.start();
} }
```

Bendras kintamasis-2. Rezultatas

Kokia tikroji d reikšmė?

1. 0?
2. 10?
3. kita?

Tarpusavio išskyrimas. Prielaidos

- “vienalaikiai” skaitymo veiksmai netrukdo vienas kitam;
- jei P ir Q vienu metu rašo į kintamąjį, tai rezultatas -— kuri nors iš rašomų reikšmių (bet ne jų junginys);
- jei P rašo, o Q -— skaito, tai Q mato arba seną, arba naują reikšmę (bet ne jų junginį).

Kritinės sekcijos apsauga

```
class P extends Thread {
  void run() {
    while (true) {
      NKS1;
      veiksmai prieš KS;
      KS;
      veiksmai už KS;
      NKS2;
    }
  }
}
```

KS apsauga. 1 variantas

```
Thread P1:  run {                | Thread P2:  run {
  while (true) {                 |   while (true) {
    NKS1;                        |     NKS1;
    while (n==2) {}              |     while (n==1) {}
    KS;                          |     KS;
    n = 2;                       |     n = 1;
    NKS2;                        |     NKS2;
  }                              |   }
}                                | }
```

Ar šis algoritmas teisingas?

KS apsauga. 2 variantas

```
Thread P1:  run {
  while (true) {
    NKS1;
    while (flag2) {}
    flag1 = true;
    KS;
    flag1 = false;
    NKS2;
  }
}
```

```
| Thread P2:  run {
|   while (true) {
|     NKS1;
|     while (flag1) {}
|     flag2 = true;
|     KS;
|     flag2 = false;
|     NKS2;
|   }
| }
```

Ar šis algoritmas teisingas?

KS apsauga. 3 variantas

```
Thread P1:  run {
  while (true) {
    NKS1;
    flag1 = true;
    while (flag2) {}
    KS;
    flag1 = false;
    NKS2;
  }
}
```

```
| Thread P2:  run {
|   while (true) {
|     NKS1;
|     flag2 = true;
|     while (flag1) {}
|     KS;
|     flag2 = false;
|     NKS2;
|   }
| }
```

Ar šis algoritmas teisingas?

Trys variantai. Išvados

1. Jei vienas procesas yra NKS, o kitas nori įeiti į KS, reikia leisti jam tai daryti.
2. Jei du procesai varžosi dėl perėjimo į KS, sprendimas, kuriam procesui leisti tai daryti, negali būti atidėtas neapibrėžtam laikui.

KS apsauga. Peterson'o algoritmas

```
Thread P1:  run {          | Thread P2:  run {
  while (true) {           |   while (true) {
    NKS1;                  |     NKS1;
    flag1 = true;          |     flag2 = true;
    n = 2;                 |     n = 1;
    while(flag2&&n==2)     |     while(flag1&&n==1)
     {}                    |      {}
    KS;                    |     KS;
    flag1 = false;         |     flag2 = false;
    NKS2;                  |     NKS2;
  }                        |   }
}                          | }
```

KS apsauga. Bakery (parduotuvės) algoritmas (a)

```
   // procesų skaičius
int n;
   // rodo, kuris procesas ima bilietą
boolean choosing [n];
   // procesų bilietų nr. (prad. reikšmės = 0)
int ticket [n];
```

KS apsauga. Bakery (parduotuvės) algoritmas (b)

```
   // Thread Pk: prieš KS:
choosing[k] = true;    // bilieto pasirinkimas
ticket[k] = maxValue(ticket) + 1;
choosing[k] = false;

for (int i=0; i<n; i++) {// laukti lyginant savo
  if (i == k) continue;  // bilietą su kt.proc.bil.
  while (choosing [i]);
  while (ticket[i]!=0 && ticket[i]<ticket[k]);
  if (ticket[i] == ticket[k] && i<k)
    while (ticket[i] != 0);
}

  // už KS:
ticket[k] = 0;
```

Sąlyginė sinchronizacija

```
// thread                          | // thread
class Siuntėjas {                  | class Gavėjas {
  int m;                           |   int d;
  void run() {                     |   void run() {
    Formuoti(m);                   |     while (!BK2.t);
    while (BK2.t);                 |     d=BK2.v;
    BK2.v=m;                       |     BK2.t=false;
    BK2.t=true;                    |     Naudoti(d);
} }                                | } }
class BK2 {
  int v; boolean t = false;
  void main() {
    ...
} }
```

Klausimai pakartojimui

1. Kas būdinga operacinių sistemų procesams?
2. Kas būdinga procesams, veikiantiems toje pačioje vartotojo programoje?
3. Ko reikia programavimo kalboje, kad galėtume kurti lygiagrečias programas?
4. Du procesai tuo pačiu metu keičia kintamojo reikšmę. Koks bus rezultatas?
5. Nusakykite Bakery kritinių sekcijų apsaugos algoritmo esmę.
6. Ar nuoseklaus programavimo konstrukcijos yra geros kritinių sekcijų apsaugos priemonės?
7. Ar nuoseklaus programavimo konstrukcijos yra geros sąlyginio sinchronizavimo priemonės?