Lygiagretusis programavimas Ada kalboje

Lygiagretus programavimas Ada kalba

Lady Augusta Ada Byron

Countess of Lovelace (1815-1852), daughter of poet Lord Byron. She was the assistant and patron of Charles Babbage; she wrote programs for his "Analytical Engine".

Ada istorija

- **1975** m. – suformuluoti reikalavimai kalbai;
- **1983** m. – ANSI Ada;
- **1995** m. – Ada 95 (pirmoji ISO standartą atitinkanti objektinio programavimo kalba)
- **2005-2007** m. – Ada 2005;
- **2012-2016** m. – Ada 2012;

- **Interneto šaltiniai**:
  - Ada Information Clearinghouse. http://www.adaic.org//
  - ACM SIGAda Home Page. http://www.sigada.org/
  - Ada Comparison Chart. http://www.adaic.org/advantages/ada-comp-chart/

Ada kalbos ypatybės

- griežtai tipizuota kalba;
- gali būti naudojami moduliai (*packages*);
- realizuotas išimčių apdorojimas;
- galimas lygiagretus procesų (*tasks*) vykdymas;
- galima kurti objektiškai orientuotas programas (Ada95).

Ada lygiagretumo modelis: Remote Invocation (RI)

- dvikryptis duomenų perdavimas;
- komunikavimo tipas: "daug-vienam" ("many-to-one");
- komunikavimas tarp procesų - be tarpininkų;
- galimas selektyvus (alternatyvus) laukimas.

Procedūros ir RI: panašumai ir skirtumai (1)

- **naudojimas**:
  - tą pačią procedūrą galima kviesti skirtingose vietose,
  - procesas gali priimti kvietimus iš kitų procesų;
- **informacijos perdavimas**:
  - kreipinys į procedūrą gali perduoti informaciją tiek į procedūrą, tiek iš procedūros,
  - *Remote Invocation* metu informaciją galima perduoti tiek procesui-priėmėjui, tiek procesui-siuntėjui;

Procedūros ir RI: panašumai ir skirtumai (2)

- **vykdymo sąlygos**:
  - procedūros veiksmai vykdomi visada, kai tik aptinkamas kreipinys,
  - RI kvietimas vykdomas tik tada, kai procesas-savininkas tai leidžia;
- **vykdytojas**:
  - procedūros veiksmai atliekami jos kvietėjo vardu,
  - kvietimo metu nurodytus veiksmus vykdo procesas-savininkas.

RI įeigos

```
entry_aprašas ::=
  entry entry_vardas[(formalūs_parametrai)] ;

entry_kreipinys ::=
  proceso_vardas.entry_vardas[(fakt_parametrai)];

accept_sakinys ::=
  accept entry_vardas[(formalūs_parametrai)] do
    sakinys;
```

Įeigų vykdymas (1)

- **`entry`** aprašų sąrašas nurodo, į kurias proceso dalis gali kreiptis kiti procesai;
- **`entry`** kreipinys kreipiasi į kito proceso atitinkamą **`accept`** sakinį;
- **`accept`** sakinys gali būti tik tarp proceso sakinių;
- **`accept`** formali dalis turi pilnai sutapti su atitinkamo **`entry`** aprašo formalia dalimi;

Įeigų vykdymas (2)

- **`accept`** parametrai matomi tik už **`accept`** užrašytame sakinyje;
- **`accept E1`** sakinys gali būti **`accept E2`** sakinio viduje;
- įeigos kreipinio faktiniai parametrai turi atitikti formaliems įeigos aprašo parametrams;
- negalima kreiptis į pasibaigusių procesų įeigas;
- į tą pačią įeigą vienu metu gali kreiptis keli procesai.

Procesų vykdymas

- procesas-kvietėjas ir procesas-vykdytojas yra sinchronizuojami;
- procesas-kvietėjas blokuojamas, kol vykdomas **`accept`** sakinys;
- procesas, bandantis vykdyti **`accept`**, yra stabdomas, jei nėra atitinkamo kreipinio;
- **`accept`** sakinys gali būti tuščias.

Programos pavyzdys (1)

```
with Text_IO; use Text_IO;
procedure sel is
   NB: constant := 10;
   task Producer;          ...
   task Consumer;          ...
   task type Buffer is     ...
begin
   Put_Line("Pagrindinė programa");
end sel;
```

Programos pavyzdys (2)

```
task Producer;
  task body Producer is
    N: Integer := 0;
  begin
    for I in 1..25 loop
      N := N + 1;
      Put("Gaminti ");
      Put(N);
      New_Line;
      Buffer.Append(N);
    end loop;
  end Producer;
```

Programos pavyzdys (3)

```
task Consumer;
  task body Consumer is
    N: Integer;
  begin
    for I in 1..25 loop
      Buffer.Take(N);
      Put("Vartoti ");
      Put(N);
      New_Line;
    end loop;
  end Consumer;
```

Programos pavyzdys (4a)

```
task type Buffer is
    entry Append(I: in Integer);
    entry Take(I: out Integer);
  end Buffer;
```

Programos pavyzdys (4b)

```
task body Buffer is
  B: array(0..NB) of Integer;
  In_Ptr, Out_Ptr:  Integer := 0;
  Count:  Integer := 0;
  begin
    loop
      ...
    end loop;
  end Buffer;
```

Programos pavyzdys (4c)

```
select
   when Count<NB =>
   accept Append(I:in Integer) do B(In_Ptr):=I;
   end Append;
   Count:=Count+1; In_Ptr:=(In_Ptr+1) mod NB;
or
   when Count>0 =>
   accept Take(I:out Integer) do I:=B(Out_Ptr);
   end Take;
   Count:=Count-1; Out_Ptr:=(Out_Ptr+1) mod NB;
or
   terminate;
end select;
```

Kuri kalba geriau tinka lygiagrečiosioms programoms kurti?

- Ada?
- C, C++?
- C + OpenMP?
- C + MPI?
- C + CUDA?
- C#?
- Go?
- Java?
- occam?
- . . . ?

Klausimai pakartojimui

1. Kuo ypatinga Ada programavimo kalba?
2. Kokiu būdu Adoje kuriami lygiagretūs procesai?
3. Kuo pasižymi Ados lygiagrečiųjų procesų komunikavimas?
4. Kuo Remote Invocation modelis panašus į procedūrų naudojimo modelį?
5. Kokiu būdu sinchronizuojami Ada procesai?