ESERCITAZIONE 5

Gestione di task concorrenti e rendez-vous in ADA

13 maggio 2024

Esempio 1: ponte a senso unico alternato

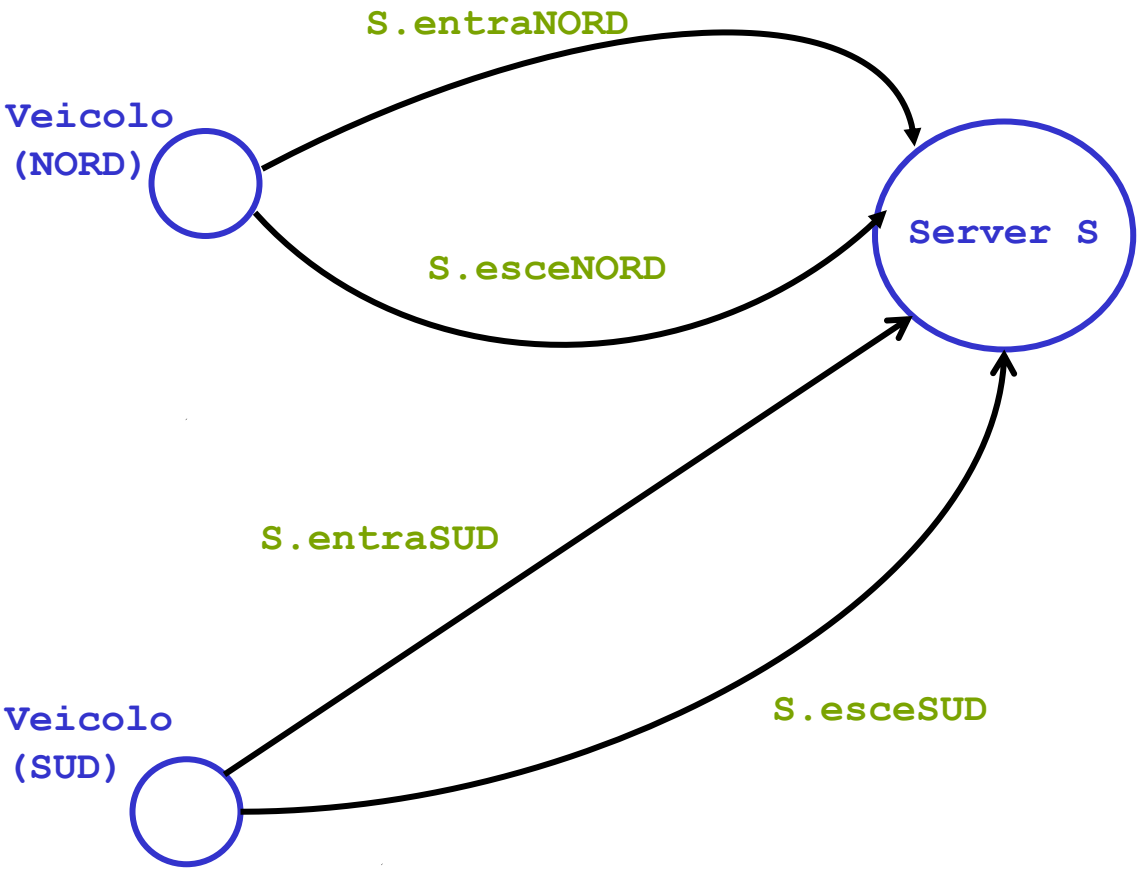
Si consideri un **ponte a senso unico alternato** con capacità limitata a MAX veicoli.

Ogni veicolo che vuole entrare dalla direzione X è autorizzato se:

- –c'è posto sul ponte (il numero di veicoli è minore di MAX)
- –non ci sono veicoli in direzione opposta a X.

Realizzare un'applicazione distribuita in ADA in cui i veicoli siano rappresentati da task concorrenti (clienti) e la gestione del ponte sia affidata ad un task (servitore).

Schema soluzione



Impostazione

```
with Ada.Text_IO, Ada.Integer_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO;

procedure ponte is
   type cliente_ID is range 1..10;  -- 10 clienti
   type dir_ID is  (NORD, SUD);     -- direzioni


   -- processo gestore del pool:
   task type server is
         entry entraNORD (ID: in cliente_ID );
         entry esceNORD(ID: in cliente_ID );
         entry entraSUD (ID: in cliente_ID );
         entry esceSUD(ID: in cliente_ID );
   end server;
  S: server;      -- creazione server
```

- **Task client**: rappresenta l'utente del ponte.

```
task type cliente (ID: cliente_ID; DIR: dir_ID);
task body cliente is
begin
Put_Line("Gruppo " & cliente_ID'Image (ID) & " dir:"& dir_ID'Image(DIR));
   if DIR=NORD
         then
         S. entraNORD(ID);
         delay 1.0;
         S. esceNORD(ID);
    end if;
    if DIR=SUD
    then
         S. entraSUD(ID);
         delay 1.0;
         S. esceSUD(ID);
    end if;
end;
```

- **Task server**: è il gestore del ponte

```
task body server   is
   MAX  : constant INTEGER := 5; --capacità ponte
   sulponte: Integer;
   utenti: array(dir_ID'Range) of Integer;

begin
   Put_Line ("SERVER iniziato!");
   --INIZIALIZZAZIONI:
    sulponte:=0;
  for i in dir_ID'Range loop
      utenti(i):=0;
    end loop;
   -- continua..
```

```
-- .. Gestione richieste
loop
        select
           when sulponte < MAX and utenti(SUD)=0  =>
           accept entraNORD (ID: in cliente_ID ) do
                      utenti(NORD):=utenti(NORD)+1;
                      sulponte:=sulponte+1;
                      end entraNORD;   -- fine sincron.
        or
           when sulponte < MAX and utenti(NORD)=0  =>
           accept entraSUD (ID: in cliente_ID ) do
                      utenti(SUD):=utenti(SUD)+1;
                      sulponte:=sulponte+1;
                      end entraSUD;
        -- continua..
```

```
            -- .. continua
            or
               accept esceNORD (ID: in cliente_ID ) do
                        utenti(NORD):=utenti(NORD)-1;
                        sulponte:=sulponte-1;
                        end esceNORD;
            or
              accept esceSUD (ID: in cliente_ID ) do
                        utenti(SUD):=utenti(SUD)-1;
                        sulponte:=sulponte-1;
                        end esceSUD;
      end select;
   end loop;
end;
```

Struttura programma e definizione main:

```
with Ada.Text_IO, Ada.Integer_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO;

procedure ponte is

-- dichiarazioni e definizioni task ecc.
...

type ac is access cliente; -- riferimento ad un task cliente
  New_client: ac;

  begin -- equivale al main
   for I in cliente_ID'Range loop  -- ciclo creazione task
      New_client := new cliente (I); -- creazione cliente I-simo
   end loop;
end ponte;
```

In alternativa: selezione entry in base a parametri

Vettore delle operazioni di servizio:

Per consentire la realizzazione di politiche dipendenti dai «**parametri**» associati al task si può usare il concetto di ***famiglie di entry***:

```
entry entryname (first..last) (in..out);
```

- Soluzione con 2 entries:

```
task type server is
      entry entra(dir_ID) (ID: in cliente_ID );
      entry esce(dir_ID)(ID: in cliente_ID );
   end server;
```

Definizione task server: struttura.

```
task body server   is
   MAX  : constant INTEGER := 5; --capacità ponte
   -- <variabili di stato del ponte>
begin
    --<inizializzaz. variabili di stato del ponte>
  loop--Gestione richieste:
       select
              ... accept entra(NORD) (ID: in cliente_ID ) do ...
       or
              ... accept entra(SUD) (ID: in cliente_ID ) do ...
       or
              ... accept esce(NORD) (ID: in cliente_ID ) do ...
       or
              ... accept esce(SUD) (ID: in cliente_ID ) do ...
       end select;
  end loop;
end;
```

Task client (con family of entries).

```
task type cliente (ID: cliente_ID; DIR: dir_ID);

task body cliente is
begin
   Put_Line ("gruppo" & cliente_ID'Image (ID) & " di "&  dir_ID'Image (DIR) &"iniziato!");
   S. entra(DIR)(ID);
   delay 1.0;
   S. esce(DIR)(ID);
end;
```

Politiche basate su priorità

Politiche basate su priorità: necessità di selezionare la richieste di entrata da servire

Vettore delle operazioni di servizio: Family of entries

Es. 3 livelli di priorità: P1, P2 e P3

- Il gestore di una risorsa deve servire prima le richieste di priorità P1, poi P2, e per ultima, P3.

```
type prio is(P1, P2, P3);

task type server is
      entry richiesta(prio) (<parametri formali>);
  ..
 end server;
```

Struttura server

Primo metodo: usare il connettore **`else`** all'interno di select (possibilità di innestare select):

```
select
        accept richiesta(P1)…
   else
        select
                accept richiesta(P2)…
        else
                select
                        accept richiesta(P3) …
                        or
                        delay 1.0;
                end select;
        end select;
end select;
```

Struttura server

Secondo metodo (consigliato): usare l'attributo `'COUNT` (applicabile alle entries).
Ad esempio:

`Richiesta(P1)'COUNT`

Restituisce **il numero di richieste in coda** per la entry Richiesta(P1).

👉 **SCHEMA da seguire:**

```
select
        accept richiesta(P1) do.. end;
   or
        when richiesta(P1)'COUNT=0 =>
        accept richiesta(P2) do.. end;
   or
        when richiesta(P1)'COUNT=0  and
             richiesta(P2)'COUNT=0 =>
        accept richiesta(P3) do.. end;
end select;
```

Esempio 2: la fabbrica di torte

Si consideri il laboratorio di un'azienda artigianale che produce dolci. L'azienda è specializzata nella produzione di torte; in particolare, i tipi di torte prodotti sono 2:

- **Torta al cioccolato**,
- **Crostata alla marmellata**.

Le torte vengono vendute in scatole pre-confezionate. L'azienda commercializza 3 tipi di confezioni:

- confezione semplice "**Cioccolato**", contenente 1 torta al cioccolato;
- confezione semplice "**Marmellata**", contenente 1 crostata.
- Confezione multipla "**Famiglia**", contenente 1 torta al cioccolato e 1 crostata.

Nel laboratorio dell'azienda vi è un **tavolo** per il deposito delle torte in attesa di essere confezionate al quale accedono:

- gli **operai dedicati alla produzione** (OP), che accedono ciclicamente al tavolo per depositarvi ogni torta appena sfornata; ogni OP deposita sul tavolo 1 torta alla volta.
- gli **operai dedicati alle confezioni** (OC), ognuno dedicato alla confezione di scatole di un tipo predefinito dato (Cioccolato, Marmellata o Famiglia); essi accedono ciclicamente al tavolo per prelevare la/le torte necessaria/e a realizzare la confezione del tipo assegnato.

Il tavolo ha una **capacità massima** pari a **MaxC**, costante che esprime il massimo numero di torte che possono stare contemporaneamente su di esso.

Si sviluppi un'applicazione distribuita ADA, che rappresenti **operai** (clienti) e **gestore** del tavolo (server) con task concorrenti. L'applicazione deve realizzare una politica di gestione del tavolo che soddisfi i vincoli dati e che, inoltre, soddisfi i seguenti vincoli di **priorità**:

- tra gli operai **OP**: **i produttori di crostate** siano **favoriti** rispetto ai **produttori di torte al cioccolato**;
- tra gli operai **OC**: gli operai dedicati alla confezione di **scatole Famiglia** siano **favoriti** rispetto a quelli dedicati alle **scatole semplici** (Cioccolato, Marmellata); inoltre, tra gli OC dedicati alle confezioni semplici, venga data priorità alle confezioni "Marmellata".

Impostazione server: famiglie di entries

```
type torta is (cioccolata, marmellata);
type confezione is (cioc, marm, family);

task type server is
 entry deposito(torta) (<par. formali>);
 entry prelievo(confezione) (<par. formali>);
end server;

S: server; -- creazione processo server
```

Impostazione clienti OP/OC

```
type cliente_ID is range 1..10;

task type clienteOP (ID: cliente_ID; T:torta);
 task body clienteOP is
  begin
          S. deposito(T)(ID);
  end;


task type clienteOC(ID: cliente_ID; C:confezione);
 task body clienteOC is
  begin
          S. prelievo(C)(ID);
  end;
```

Politica del gestore

E' realizzata all'interno del server:

```
task body server   is
   <variabili locali per rappr. Stato risorsa>
   begin
  <INIZIALIZZAZIONI>
   loop
  select
       <accettazione/definizione entries>
  end select:
   end loop;
end server;
```

Soluzione Completa

```
with Ada.Text_IO, Ada.Integer_Text_IO;
use Ada.Text_IO, Ada.Integer_Text_IO;

procedure torte is
   type clienteOP_ID is range 1..10;
   type clienteOC_ID is range 1..4;
   type torta is (cioccolato, marmellata);
   type confezione is (cioc, marm, family);
   task type clienteOP (ID: clienteOP_ID; T:torta);
   task type clienteOC (ID: clienteOC_ID; C:confezione);
   type acOP is access clienteOP;
   type acOC is access clienteOC;

   task type server is
         entry deposito(torta) (ID:clienteOP_ID);
         entry prelievo(confezione) (ID:clienteOC_ID);
   end server;
```

```
S: server;

task body server   is
   MAX  : constant INTEGER := 18; -- capacita' tavolo

   sultavolo: array(torta'Range) of Integer;
   begin
   Put_Line ("SERVER iniziato!");
   --INIZIALIZZAZIONI:
   for i in torta'Range loop
      sultavolo(i):=0;
   end loop;
   delay 2.0;

        -- continua..
```

```
loop
  select  -- deposito crostata:
    when  sultavolo(marmellata)+sultavolo(cioccolato)<MAX  and
    sultavolo(marmellata) < MAX-1 =>
    accept deposito(marmellata) (ID: in clienteOP_ID ) do
    sultavolo(marmellata):=sultavolo(marmellata)+1;
    end;
  or   -- deposito cioccolato:
    when sultavolo(marmellata)+sultavolo(cioccolato)<MAX and
    sultavolo(cioccolato) < MAX-1 and
    deposito(marmellata)'COUNT=0 =>
    accept deposito(cioccolato) (ID: in clienteOP_ID ) do
    sultavolo(cioccolato):=sultavolo(cioccolato)+1;
  end;
        -- CONTINUA..
```

```
or  -- prelievo family:
  when sultavolo(marmellata) >=1
  and sultavolo(cioccolato) >=1 =>
  accept prelievo(family) (ID: in clienteOC_ID ) do
  sultavolo(marmellata):=sultavolo(marmellata)-1;
  sultavolo(cioccolato):=sultavolo(cioccolato)-1;
  end;
or -- prelievo marmellata:
  when sultavolo(marmellata) >=1 and prelievo(family)'COUNT=0 =>
  accept prelievo(marm) (ID: in clienteOC_ID ) do
  sultavolo(marmellata):=sultavolo(marmellata)-1;
  end;
or -- prelievo cioccolato
  when sultavolo(cioccolato) >=1 and prelievo(family)'COUNT=0
  and prelievo(marm)'COUNT=0=>
  accept prelievo(cioc) (ID: in clienteOC_ID ) do
  sultavolo(cioccolato):=sultavolo(cioccolato)-1;
  end;
 end select;
end loop;
end; -- fine task server
```

```
-- definizione task clienti:

task body clienteOP is
begin
        S. deposito(T)(ID);
end;


task body clienteOC is
begin
        S. prelievo(C)(ID);
end;
```

```
-- "main":
NewOP: acOP;
NewOC: acOC;

begin -- equivale al main
   for I in clienteOP_ID'Range
   loop  -- ciclo creazione task OP
      NewOP := new clienteOP (I, cioccolato);
        NewOP := new clienteOP (I, marmellata);
   end loop;

   for I in clienteOC_ID'Range
   loop  -- ciclo creazione task OC
          NewOC := new clienteOC (I, cioc);
          NewOC := new clienteOC (I, marm);
          NewOC := new clienteOC (I, family);
        end loop;
end torte; -- fine programma
```

Risorse utili

- Compilatore linux: **gnat**
- Comando per compilazione:

```
gnat make programma.adb
```

- Ambiente grafico di sviluppo:

Gnat Programming Studio (GPS)

- Per download GPS etc:

```
http://www.adacore.com
```

- Tutorial ADA on line:

http://www.adaic.org/learn/materials/#tutorials

http://www.infres.enst.fr/~pautet/Ada95/a95list.htm

Esercizi proposti

Esercizio 1

Si consideri l'ufficio di relazioni con il pubblico (**URP**) di una grande città.
L'ufficio è costituito da N sportelli, attraverso i quali è in grado di fornire al pubblico 2 tipi di prestazione:

- Informazioni turistiche (**TUR**)
- Informazioni su eventi (**EVE**)

Ogni sportello può eseguire un servizio alla volta (di qualunque tipo).
Per semplicità, si assuma che ogni utente richieda un solo servizio alla volta.
Si assuma inoltre che la permanenza di un utente allo sportello abbia una durata non trascurabile.
L'accesso degli utenti agli sportelli è regolato dai seguenti vincoli:

- l'erogazione di un servizio a un utente presuppone l'acquisizione di uno sportello libero da parte dell'utente richiedente.

Realizzare un'applicazione nel linguaggio **Ada**, nella quale **utenti e ufficio** siano rappresentati **task concorrenti**.

La sincronizzazione tra i processi dovrà tenere conto dei vincoli dati ed inoltre della seguente politica di priorità:

le richieste di **informazioni turistiche (TUR)** devono avere la **precedenza sulle richieste di tipo EVE**.

Impostazione

- **Due tipi di task:**
  - Utenti (clienti)
  - Ufficio (server)

- **Quali servizi?**
  - Acquisizione sportello (EVE/TUR)
  - Rilascio Sportello

- **Gestione delle Priorità (nell'acquisizione):**
  - Acquisizione -> Famiglia di entries (EVE, TUR)
  - Uso dell'attributo 'COUNT applicato ai due tipi di entry per verificare la presenza di chiamate più prioritarie

Impostazione

Cliente:

```
task type cliente(id: ...; tipo_info:...);
task body cliente is
begin
        S.acquisizione(tipo_info)(id,..);
        <permanenza allo sportello>
        S.Rilascio(id,..);
end;
```

Esercizio 2

Si realizzi una variante della soluzione dell'esercizio 1, in cui la politica di priorità sia la seguente.

Riguardo all'ordine delle richieste servite, si adotti un criterio basato su **priorità dinamica** e cioè:

1)Inizialmente la priorità è assegnata alle richieste di tipo **TUR**;

2) **dopo aver servito K richieste TUR** (K è una costante data), la priorità viene invertita, quindi diventano prioritarie le richieste di tipo **EVE**.

3) Analogamente, **dopo aver servito K richieste** di tipo **EVE**, la priorità viene ancora invertita, e verrà data la precedenza a richieste di tipo **TUR**, e così via , ricominciando dal punto 2.

Realizzare un'applicazione nel linguaggio **Ada**, nella quale **utenti e ufficio** siano rappresentati **task concorrenti**.
La sincronizzazione tra i processi dovrà tenere conto dei vincoli dati.