ESERCITAZIONE 5

27 novembre 2017

Programmazione concorrente in ADA

Risorse utili

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

```
gnat make programma.adb
```

- Per download pugin ADA per eclipse, xcode, etc:

`http://www.adacore.com`

- Tutorial ADA on line:

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

Esempio: 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 portatori di crostate siano favoriti rispetto ai portatori 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;
```

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

- 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
```

Esercizio 1

Si consideri un ponte a **senso unico alternato** con capacità limitata a MAX veicoli. I veicoli possono essere di 2 tipi:

- **Camion**
- **Auto**

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 ADA in cui i veicoli (auto o camion) siano rappresentati da task concorrenti e la gestione del ponte sia affidata ad un task servitore. Nell'accesso al ponte, si dia la **precedenza alle auto**.

Esercizio 2

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.

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, 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 1.

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