

# ESERCITAZIONE 5

5 dicembre 2013

# 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: pool di risorse equivalenti

- 3 tipi di task:
  - **Server**: gestore del pool di risorse
  - **Risorsa**: gestore di una risorsa del pool
  - **Cliente(id)**: task che usa le risorse del pool

## DICHIARAZIONI:

```
with Ada.Text_IO; use Ada.Text_IO;
procedure pool1 is

type cliente_ID is range 1..10;           -- 10 clienti
type ris_ID is range 1..5;                -- 5 risorse nel pool

task type cliente (ID: cliente_ID); -- task cliente
type ac is access cliente;           -- riferimento ad un cliente

task type risorsa (ID: ris_ID) is      -- task gestore di 1 risorsa
    entry operazione(ID: in cliente_ID; OK: out Boolean);
end risorsa;

type ar is access all risorsa; -- rif. ad un task gestore di risorsa

type pool is array(ris_ID'Range) of ar;

task type server is -- processo gestore del pool
    entry Richiesta (ID: in cliente_ID; RIS: out ar; IND: out ris_ID);
    entry Rilascio(ID: in cliente_ID; IND: in ris_ID);
end server;
```

# Definizione Cliente:

P: pool; --creazione pool

S: server; -- creazione server

```
task body cliente is
```

```
    GR: ar;
```

```
    INDEX: ris_ID;
```

```
    OK: Boolean;
```

```
begin
```

```
    Put_Line ("cliente" & cliente_ID'Image (ID) & " !");
```

```
    S. Richiesta(ID, GR, INDEX);
```

```
    Put_Line ("cliente ha ottenuto una risorsa!");
```

```
    GR.operazione(ID, OK); ---Accesso alla risorsa
```

```
    S.Rilascio(ID, INDEX);
```

```
        Put_Line ("cliente terminato!");
```

```
end;
```

## Definizione server:

```
task body server    is    -- definizione PROCESSO
server
    disp: Integer;
    k: ris_ID;
    libere: array(ris_ID'Range) of Boolean;
begin
    k:=1;
    Put_Line ("SERVER iniziato!");
    -- inizializzazione stato server:
    disp:=5;
    for i in ris_ID'Range loop
        libere(i):=True;
    end loop;
    -- continua
```

```

loop
select
  when disp > 0 =>  -- c'è almeno una risorsa libera
  accept Richiesta (ID: in cliente_ID; RIS: out ar; IND: out
ris_ID)
  do
    Put_Line ("server ha ricevuto richiesta dal cliente
"& cliente_ID'Image(ID) &" !");
    for k in ris_ID'Range loop
      if libere(k)=True
      then
        libere(k):=False;
        disp:=disp-1;
        IND:=k;
        RIS :=P(k);
        exit;
      end if;
    end loop;
  end Richiesta;          -- fine servizio
or

```



```
accept  Rilascio(ID: in cliente_ID; IND: in  
ris_ID)  
do  
    libere(IND) := True;  
    disp := disp + 1;  
end Rilascio; -- fine servizio  
  
end select;  
end loop;  
end; -- fine server
```

## Def. risorsa

```
task body risorsa is
  miod: ris_ID;
begin
  Put_Line ("risorsa " & ris_ID'Image (ID) & " ! ");
  miod:=ID;
  loop
    select
      accept operazione (ID: in cliente_ID; OK: out
        Boolean) do
        Put_Line ("risorsa " & ris_ID'Image (miod) & " in
          uso da " & cliente_ID'Image (ID) & " ! ");
        delay 1.0;
        OK:=True;
      end operazione;
    end select;
  end loop;
end risorsa;
```

## Def. "main"

```
New_client: ac;
```

```
begin -- equivale al main
```

```
    for I in ris_ID'Range loop
```

```
        P(I):=new risorsa(I);
```

```
    end loop;
```

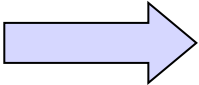
```
for I in cliente_ID'Range loop -- creaz. task
```

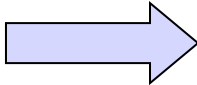
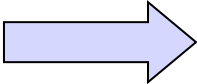
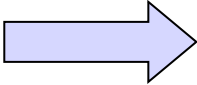
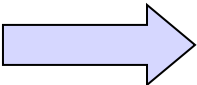
```
    New_client := new cliente (I);
```

```
end loop;
```

```
end pool1; -- fine programma
```

# Corrispondenza tra monitor e processi servitori in ADA

<b>modello a memoria comune</b>	<b>corrisponde</b>	<b>modello a scambio di messaggi (ADA)</b>
risorsa condivisa: istanza di un monitor		risorsa condivisa: struttura dati locale a un processo server
identificatore di funzione di accesso al monitor		nome di <b>entry</b> offerta dal server
tipo dei parametri della funzione		tipo dei parametri <b>in</b> della entry
tipo del valore restituito dalla funzione		tipo parametri <b>out</b> della entry
per ogni funzione del monitor		un ramo ( <b>comando con guardia</b> ) dell'istruzione <b>loop-select</b> che costituisce il corpo del server

<b>modello a memoria comune</b>	<b>corrisponde</b>	<b>modello a scambio di messaggi (ADA)</b>
condizione di sincronizzazione di una funzione		espressione logica ( <b>when</b> ) nel ramo corrispondente alla funzione
chiamata di funzione		chiamata (da parte del client) della entry corrispondente nel server
esecuzione in mutua esclusione fra le chiamate alle funzioni del monitor		scelta di uno dei rami con guardia valida del loop-select del server
corpo della funzione		istruzione del ramo corrispondente alla funzione

## Esercizio

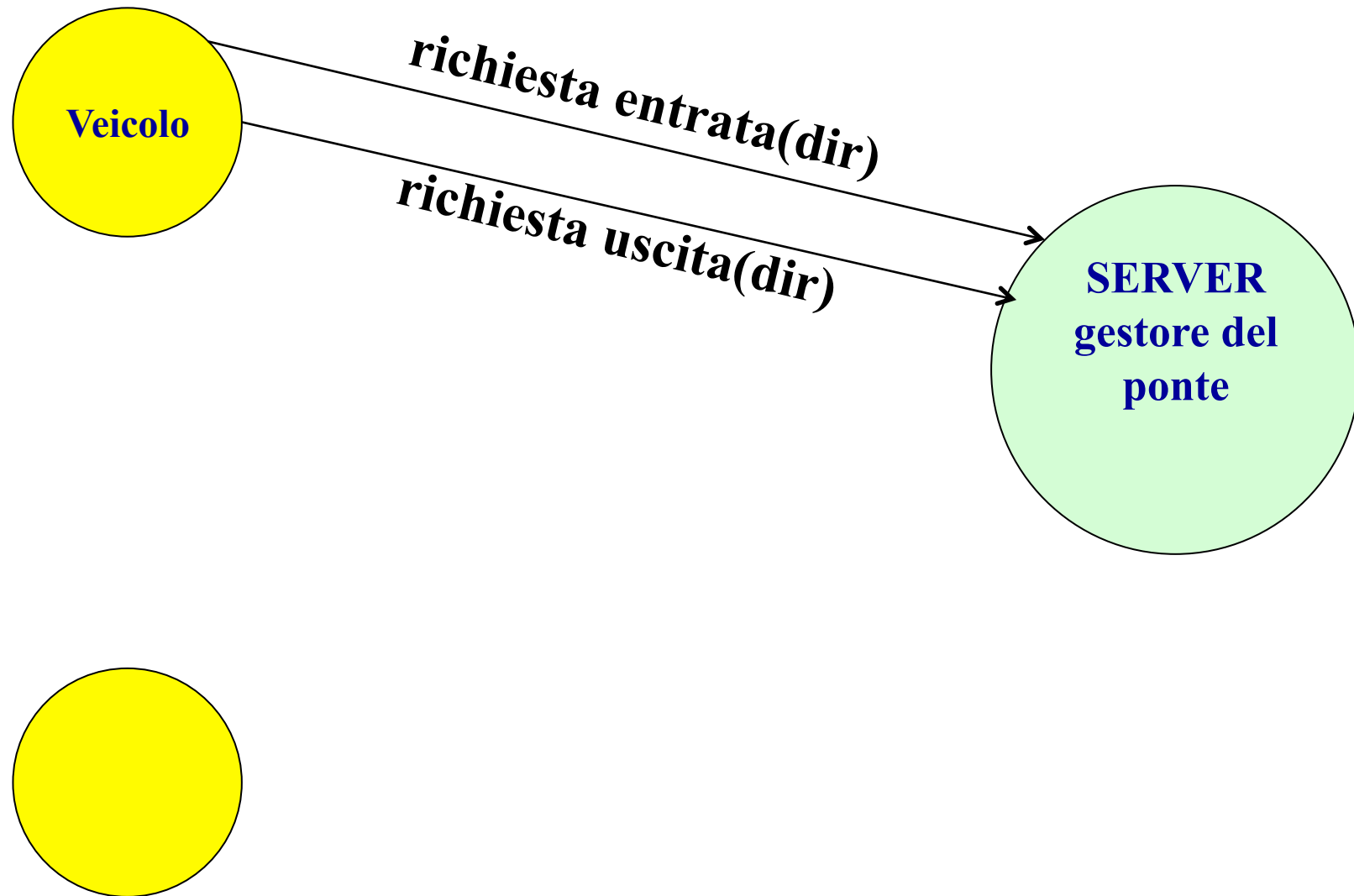
Si consideri un ponte a senso unico con capacità limitata a  $MAX$  veicoli.

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 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

- Task server per la gestione degli accessi/uscite al ponte:

```
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;

server S;
```



- 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) & " di "&
              dir_ID'Image (DIR) & "iniziato!");
    loop
        if DIR=NORD
        then
            S. entraNORD(ID);
            delay 1.0;
            S. esceNORD(ID);
            delay 1.0;
        end if;
        if DIR=SUD
        then
            S. entraSUD(ID);
            delay 1.0;
            S. esceSUD(ID);
            delay 1.0;
        end if;
    end loop;
end;
```

- Definizione task server: struttura.

```
task body server is
    MAX : constant INTEGER := 5; --capacità ponte
    -- <inserire variabili di stato del ponte>
Begin
    --<inizializzare variabili di stato del ponte>
    --Gestione richieste:
    loop
        select
            --<accettazione richieste di accesso da nord>
        or
            --<accettazione richieste di accesso da sud>
        or
            --<accettazione richieste di uscita da nord>
        or
            --<accettazione richieste di uscita da sud>
        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:  
Family of entries
- 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
    -- <inserire variabili di stato del ponte>
begin
    --<inizializzare 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;
```



## Esercizio 2

Si risolva l'esercizio 1 con la seguente variante:

I veicoli possono essere di 2 tipi:

- Camion
- Auto

Nell'accesso al ponte, si dia la precedenza alle auto.