Lançamento dinâmico de tarefas

- **Nem sempre as tarefas têm de estar activas.**
- **Ou seja, uma determinada tarefa pode apenas ocorrer quando determinadas circunstâncias se verificarem.**
- **Como lançar dinamicamente uma terefa?**

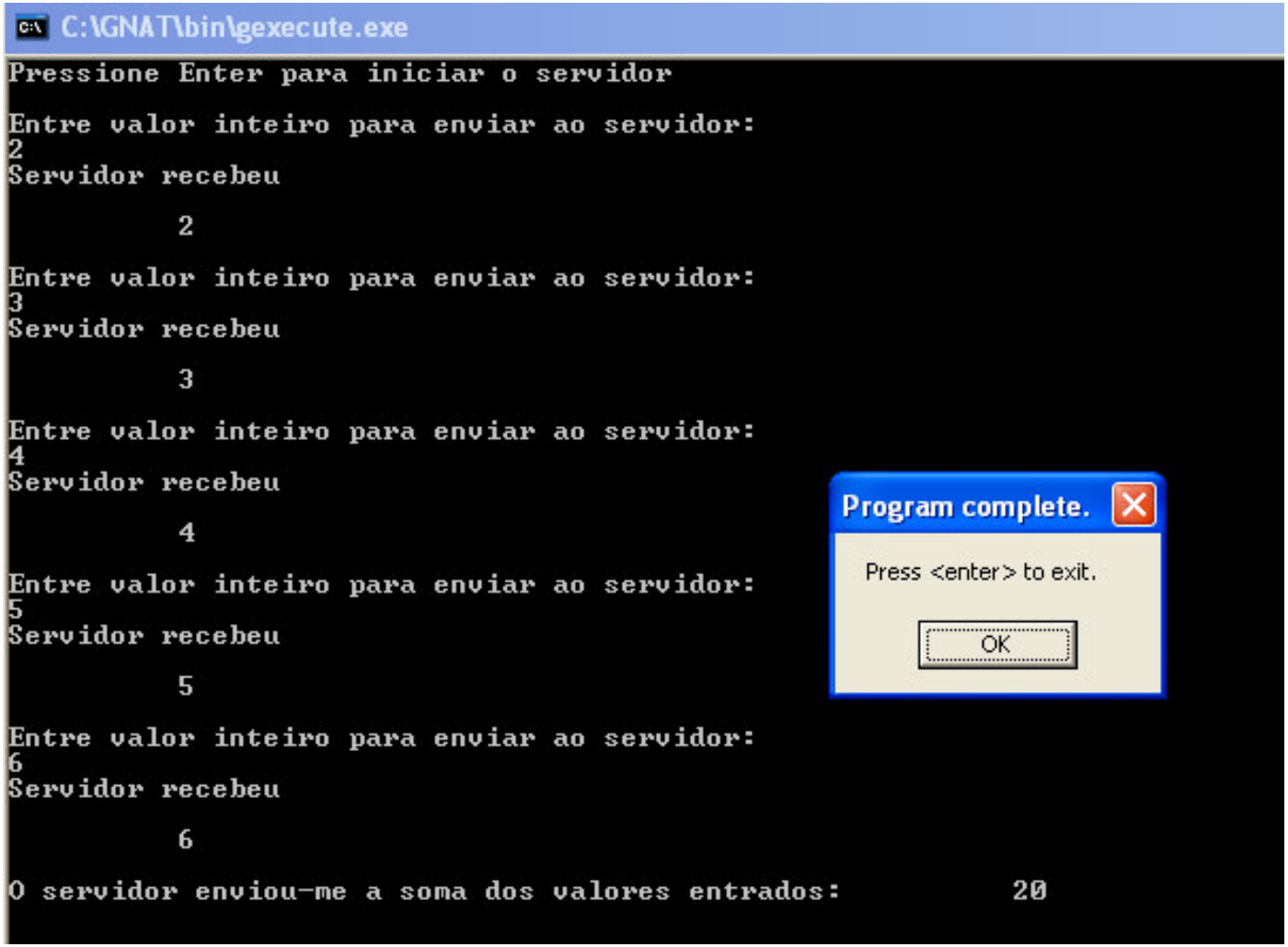
```
with Ada.Text_Io, Ada.Integer_Text_Io;
use Ada.Text_Io, Ada.Integer_Text_Io;

procedure Mais_Cachorros is

  task Cozinheiro is
    entry Faz_Cachorro(Com_Mostarda : Boolean);
  end Cozinheiro;

  task body Cozinheiro is
  begin
    Put_Line("Estou pronto para receber um pedido");
    for Index in 1..5 loop
      accept Faz_Cachorro(Com_Mostarda : Boolean) do
        Put("Cachorro ");

        Put(" esta a ser feito ");
        if com_mostarda then
          Put_Line("estou a adicionar mostarda");
        else
          Put_Line("sem mostarda");
        end if;
        delay 0.8;
      end Faz_Cachorro;
    end loop;
    Put_Line("Não tenho mais cachorros");
  end cozinheiro;

task type Pedro;
task type Joao;
```

```
   task body Pedro is
  begin
       Cozinheiro.Faz_Cachorro(False);
       Put_Line("O pedro esta a comer o cachorro sem
mostarda");
       New_Line;
  end Pedro;

  task body Joao is
  begin
       Cozinheiro.Faz_Cachorro(True);
       Put_Line("O joao esta a comer o seu cachorro
com mostarda");
       New_Line;
  end Joao;

  Type tarefa_pedro is access pedro;
  Type tarefa_joao is access joao;

  tP:tarefa_pedro;
  tJ:tarefa_joao;

begin
for i in 1..5 loop
new_line(3);
put_line("lancamento de pedro");
tp:= new pedro;
delay 3.0;
put_line("lancamento de joao");
tJ:= new joao;
delay 3.0;
end loop;
end Mais_Cachorros;
```

Servidor/cliente

```
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Integer_Text_IO;
use Ada.Integer_Text_IO;
procedure Tarefas_Rendezvous2 is

  -- tarefa servidora
  task Servidor is
    entry Start; -- Declaração do ponto de ligação
    entry Serviço1(Result: out Integer);
    entry Serviço2(dados: Integer);
  end Servidor;

  -- Corpo da tarefa servidora
  task body Servidor is
    Valor: Integer := 0;
  begin
    accept Start; -- Aqui o servidor fica bloqueado à espera
    for Count in 1..5 loop
      accept Serviço2(dados: Integer) do
        Put(Item => "Servidor recebeu ");
        Put(dados);
        New_Line;
        Valor:= Valor + dados;
      end Serviço2;
    end loop;
    accept Serviço1(Result: out Integer) do
      Result:= Valor;
    end Serviço1;
  end Servidor;
```

```
  -- tarefa cliente
  task Cliente;

  -- Corpo da tarefa cliente
  task body Cliente is
    Val: Integer;
  begin
    Put_Line("Pressione Enter para iniciar o servidor");
    Skip_Line;
    Servidor.Start; -- Aqui o servidor fica bloqueado à espera
    for Count in 1..5 loop
      Put_Line("Entre valor inteiro para enviar ao servidor: ");
      Get(Val);
      Servidor.Serviço2(Val);
    end loop;
    Servidor.Serviço1(Val);
    Put("O servidor enviou-me a soma dos valores entrados: ");
    Put(Val);
    New_Line;
  end Cliente;

begin -- Inicio da tarefa principal
  null;
end Tarefas_Rendezvous2;
```


C:\GNAT\bin\gexecute.exe
Pressione Enter para iniciar o servidor
Entre valor inteiro para enviar ao servidor:
2
Servidor recebeu
2
Entre valor inteiro para enviar ao servidor:
3
Servidor recebeu
3
Entre valor inteiro para enviar ao servidor:
4
Servidor recebeu
4
Entre valor inteiro para enviar ao servidor:
5
Servidor recebeu
5
Entre valor inteiro para enviar ao servidor:
6
Servidor recebeu
6
O servidor enviou-me a soma dos valores entrados: 20
Program complete.
Press <enter> to exit.
OK

Tarefas periódicas

```
-- Este programa cria uma tarefa periódica infinita
-- Utilizando o tipo Ada.Real_Time.Time

with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Real_Time;
use Ada.Real_Time;
```

```
procedure Tarefa_Periodica is

  task periodica;

  task body periodica is
    Period: Time_Span := Milliseconds(3000);
    Start: Time;
  begin
    Start:=Clock;
    loop
      Put_Line("Aqui estou eu. A tarefa periódica");
      Start := Start + Period;   -- O cálculo do no instante de início
      delay until Start;         -- Fica suspensa até ao instante calculado
    end loop;
  end periodica;

begin
  null;
end Tarefa_periodica;
```

- Objectos protegidos
  - São um mecanismo de alto nível para comunicação, sincronização e exclusão entre tarefas
  - Podemos declarar um objecto ou um tipo de objectos
  - Tal como as tarefas também podem ter discriminantes
  - Baseiam-se em conceitos de Orientado a Objectos
  - Um objecto protegido contém dados, e subprogramas
    - Os subprogramas podem ser públicos ou privados
  - Existem regras no acesso aos subprogramas:
    - **Procedimento**s são para leitura/escrita. Quando uma tarefa está a aceder a um objecto através de um procedimento, sabemos que mais nenhuma tarefa está a aceder ao objecto
    - **Funções** são apenas para leitura. Podem várias tarefas ao mesmo tempo acederem ao objecto para leitura, desde que nenhuma para escrita
    - **Entrie**s também são para leitura/escrita, mas no entanto têm condições adicionais (mais à frente).

```
Declaração ::=
protected [type] nome [Discriminantes] is
        {protected_operation}
[private
        {protected_element}]
end [nome];

protected_operation::= entry|procedure|function|pragma
protected_element::= variável|protected_operation

Corpo::=
protected body nome is
        Implementação das operações
end [nome];
```

```
-- Este programa cria uma tarefa periódica
-- Utilizando o tipo Ada.Real_Time.Time

with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Integer_Text_IO;
use Ada.Integer_Text_IO;

procedure Objecto_Protegido is

  protected obj_prot is
    procedure escreve(D: Integer);
    procedure reset;
    function le return Integer;
  private
    Valor: Integer := 0;
  end obj_prot;

  protected body obj_prot is
    procedure escreve(D: Integer) is
    begin
      Valor := D;
    end Escreve;
    procedure reset is
    begin
      Valor := 0;
    end Reset;
    function le return Integer is
    begin
      return Valor;
    end le;
  end obj_prot;

  task consumidor;
  task produtor;

  task body consumidor is
    begin
      for I in 1..2 loop
        delay 1.0;
        Put("O valor é ");
        Put(obj_prot.le);
        New_Line;
      end loop;
    end consumidor;

    task body produtor is
    begin
      obj_prot.escreve(2);
      delay 1.0;
      obj_prot.reset;
    end produtor;

begin
  null;
end Objecto_Protegido;
```

- Neste programa tivemos que inserir delays para garantir que só liamos após uma escrita
  - Esta solução não é muito boa
  - Para isso existem as entries

Objectos protegidos (entries)

- Numa entry é possível definir uma condição de entrada no objecto protegido
- Uma tarefa pode ficar bloqueada à espera que uma determinada condição fique verdadeira
- Essa condição pode ser activada por outra tarefa que modifica algo, ou pelo próprio runtime, quando detecta alguma condição (por exemplo mais tarefas à espera do mesmo serviço)
- Para resolver o problema anterior poderíamos modificar o objecto protegido para apenas permitir leitura após uma escrita e vice-versa.

```
protected obj_prot is
   entry escreve(D: Integer);
   entry le(Val: out Integer);
 private
   Valor: Integer := 0;
   Escrita: Boolean := false;
end obj_prot;

protected body obj_prot is
   entry escreve(D: Integer) when Escrita = false is
   begin
     Valor := D;
     Escrita := true;
   end Escreve;

   entry le(Val: out Integer) when Escrita = true is
   begin
     Val := Valor;
     Escrita := false;
   end le;
end obj_prot;
```

Objectos protegidos (entries)

- Uma tarefa bloqueada numa entry não impede o acesso de outras ao objecto
- Existe uma fila de espera por cada entry, além da fila de espera do próprio objecto (para acesso ao objecto como um todo)
- Quando uma tarefa tenta entrar numa entry primeiro tem que ganhar acesso ao objecto (mais nenhuma tarefa pode estar a executar no objecto)
  - Depois é avaliada a condição da entry
  - Se esta for verdadeira a entry comporta-se como um procedimento
  - Se for falsa a tarefa é colocada na fila de espera da entry, e liberta o acesso ao objecto
  - Quando algo modifica o estado do objecto (uma tarefa executou um procedimento no objecto, ou outra tarefa ficou bloqueada na mesma ou noutra entry) as condições de bloqueio são reavaliadas
  - É possível a condição de bloqueio serem várias condições, e dependerem do número de tarefas bloqueadas numa entry (através do atributo *'Count*)
- Para permitir o modelo Ceiling Locking existem restrições num objecto protegido
  - Num subprograma de um objecto protegido não se pode executar instruções que possam bloquear a tarefa
    - Rendezvous, acesso a uma entry de outro objecto, delay, etc..

```
-- Este programa sincroniza três tarefas
-- através de um protected object

with Ada.Text_IO;
use Ada.Text_IO;

procedure Sincro_obj is

  protected obj_prot is
    entry continua(Id_Tarefa: Character);
  private
    libertado: Boolean := false;
  end obj_prot;

  protected body obj_prot is
    entry continua(Id_Tarefa: Character) when continua'Count = 3 or libertado = true is
    begin
      if continua'Count = 2 then
        Put_Line("Eu sou a tarefa " & Id_Tarefa &
                 ". Estamos as três no objecto, portanto posso libertar todas");
        libertado := true;
      elsif continua'Count = 0 then
        Put_Line("Eu sou a tarefa " & Id_Tarefa &
                 ". Sou a ultima a sair, portanto vou voltar a fechar o objecto");
        libertado := false;
      end if;
    end continua;
  end obj_prot;
```

```
  task type tarefa(Id: Character; Espera: Integer);
  task body tarefa is
  begin
    delay Duration(Espera);
    Put_Line(Id & " vai esperar pelas outras");
    obj_prot.continua(Id);
    Put_Line("Tarefa " & Id & " já saiu do objecto");
  end tarefa;

  ta: tarefa('A', 1);    tb: tarefa('B', 2);    tc: tarefa('C', 3);

begin
  null;
end Sincro_obj;
```

Alguns erros comuns

Operator	C/C++	Ada
Assignment	=	: =
Equality	==	=
NonEquality	! =	/ =
Modulus	%	mod
AbsoluteValue		abs
Exponentiation		**
Range		..

C++

```
int i;
int a, b, c;
int j = 0;
int k, l = 1;
```

l fica com 1

k fica indefinido

Ada

```
i : Integer;
a, b, c : Integer;
j : Integer := 0;
k, l : Integer := 1;
```

l e k fica com 1

C++

```
const int days_per_week = 7;
```

```
typedef int INT;
INT a;
int b;
```

a = b → sem problemas

a ≠ A

Ada

```
days_per_week : constant Integer := 7;
days_per_week : constant := 7;
```

Em Ada, uma constante pode ser definida com um tipo ou deixar ao compilador a definição do tipo

```
type INT is new Integer;
a : INT;
b : Integer;
```

a = b → Falha

a = A

```
subtype INT is Integer;
a : INT;
b : Integer;
```

a = b → Funciona

Restringir tipos

C++

```
struct _device {
  int   major_number;
  int   minor_number;
  char  name[20];
};
typedef struct _device Device;
```

```
Device lp1 = {1, 2, "lp1"};
```

Ada

```
type BYTE is mod 256;
type BYTE is mod 2**8;
```

```
type struct_device is
  record
    major_number : Integer;
    minor_number : Integer;
    name         : String(1 .. 19);
  end record;
type Device is new struct_device;
```

```
lp1 : Device := (1, 2, "lp1");
```

C++

```
int test[5] = { 1, 2, 3, 4, 5 };
```

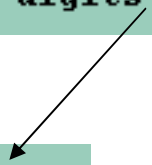
test[0] = 1, test[1] = 2, test[2] = 3, test[3] = 4, and test[4] = 5.

Ada

```
Test: array (1..5) of integer: = ( 1, 2, 3, 4, 5 };
```

Test(1) = 1, test(2) = 2, test(3) = 3, test(4) = 4, and test(5) = 5.

```
type FloatingPoint1 is new Float;
type FloatingPoint2 is digits 5;
```



```
number_of_digits : Integer := FloatingPoint2'Digits;
```

```
type Fixed is delta 0.1 range -1.0 .. 1.0;
```

```
type Decimal is delta 0.01 digits 10;
```

-99 999 999.99 a 99 999 999.99

```
type Hours   is new Integer range 1 .. 12;
type Hours24 is range 0 .. 23;
type Minutes is range 1 .. 60;
```

C++

```
char name[31];
int  track[3];
int  dbla[3][10];
int  init[3] = { 0, 1, 2 };
typedef char[31] name_type;
track[2] = 1;
dbla[0][3] = 2;
```

Ada

```
Name  : array (0 .. 30) of Character; -- OR
Name  : String (1 .. 30);
Track : array (0 .. 2) of Integer;
```

```
DblA  : array (0 .. 2,0 .. 9) of Integer;
Init  : array (0 .. 2) of Integer := (0, 1, 2).
type Name_Type is array (0 .. 30) of Character;
track(2) := 1;
dbla(0,3) := 2;
```