18 - Tâches et exceptions

Programmation Concurrente - LI330
Université P. & M. Curie - année scolaire 2013/2014

PrC

Quelques principes

- La notion d'exception est liée à un «flux d'exécution»
  - Programme séquentiel : exception non rattrapée = fin du programme
  - Programme parallèle : exception non rattrapée = fin de la tâche qui l'a levée
    - Mais pas forcément du programme!
- Les exceptions peuvent donc être rattrapées au niveau d'une tâche
  - Elles peuvent aussi être rattrapées dans le traitement d'un point d'entrée...

    ... et par extension dans les fonctions/procédures des objets protégés
- Les exceptions peuvent se propager entre tâches ou objets protégés
  - Seul cas de figure possible : levée dans le traitement...
    - ...d'un point d'entrée (tâches et objets protégés)
    - ...d'une procédure ou fonction (objets protégés)

Exemple 1: comportement (1)

```
with Ada.Text_IO, Ada.Exceptions;
   use Ada.Text_IO, Ada;
   procedure Example_Exception_Tache_1 is
      task T1;
      task T2;
      task body T1 is
         A, B : Natural := 2;
      begin
         Put_Line ("Debut du calcul");
         delay 1.0;
         A := A - 2;
         B := B / A; -- lève Constraint_Error
         Put_Line ("Fin du calcul");
      exception
         when E : others =>
            Put_Line ("T1 meurt sur " & Exceptions.Exception_Name
   (E));
      end T1;
   ...
```

Exemple 1: comportement (2)


```
...
   task body T2 is
   begin
      loop
         Put_Line ("   T2 dit
coucou");
         delay 1.0;
      end loop;
   end T2;
begin -- démarrage de T1 et T2...
   null;
end Example_Exception_Tache_1;
```

Exemple 1: et si l'exception concerne la tâche d'environnement.

Changeons le corps du programme Principal

```
begin
   raise Probleme;
exception
   when E : others =>
      Put_Line ("'main' meurt sur " &
                Exceptions.Exception_Name (E));
end Example_Exception_Tache_1bis;
```

Exemple 2: propagation entre tâches (1)

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

   procedure Example_Exception_Tache_2 is

      Probleme : exception;

      task type T1 is
         entry Init (Id : in Natural);
         entry Service;
      end T1;

      Tab : array (1 .. 10) of T1;
   ...
```

Exemple 2: propagation entre tâches (2)


```
...
   task body T1 is
      Myself : Natural;
   begin
      accept Init (Id : in Natural) do
         Myself := Id;
      end Init;
      Put_Line ((1 .. Myself => ' ') & "T1 avec" &
                Natural'Image (Myself) & " est initialise");
      accept Service do
         Put_Line ((1 .. Myself => ' ') & "T1 avec" &
                   Natural'Image (Myself) & " traite un service");
         if Myself = Tab'Last then
            raise Probleme;
         else
            Tab (Myself + 1).Service;
         end if;
      end Service;
   exception
      when E : others =>
         Put_Line ((1 .. Myself => ' ') & "T1 avec" & Natural'Image
(Myself) &
                   " meurt sur " & Exceptions.Exception Name (E));
```

Exemple 2: propagation entre tâches (3)


```
...
   begin
      for I in  Tab'Range loop
         Tab (I).Init (I);
      end loop;
      Tab (Tab'First).Service;
   end
   Example_Exception_Tache_2;
```

Exemple 2: propagation entre tâches (4)


```
T1 avec 1 traite un service
 T1 avec 2 traite un service
  T1 avec 3 traite un service
   T1 avec 4 traite un service
    T1 avec 5 traite un service
     T1 avec 6 traite un service
      T1 avec 7 traite un service
       T1 avec 8 traite un service
        T1 avec 9 traite un service
         T1 avec 10 traite un service
         T1 avec 10 meurt sur EXAMPLE_EXCEPTION_TACHE_2.PROBLEME
        T1 avec 9 meurt sur EXAMPLE_EXCEPTION_TACHE_2.PROBLEME
       T1 avec 8 meurt sur EXAMPLE_EXCEPTION_TACHE_2.PROBLEME
      T1 avec 7 meurt sur EXAMPLE_EXCEPTION_TACHE_2.PROBLEME
     T1 avec 6 meurt sur EXAMPLE_EXCEPTION_TACHE_2.PROBLEME
   T1 avec 4 meurt sur EXAMPLE_EXCEPTION_TACHE_2.PROBLEME
    T1 avec 5 meurt sur EXAMPLE_EXCEPTION_TACHE_2.PROBLEME
  T1 avec 3 meurt sur EXAMPLE_EXCEPTION_TACHE_2.PROBLEME
 T1 avec 2 meurt sur EXAMPLE_EXCEPTION_TACHE_2.PROBLEME
T1 avec 1 meurt sur EXAMPLE_EXCEPTION_TACHE_2.PROBLEME

raised EXAMPLE_EXCEPTION_TACHE_2.PROBLEME : example_exception_tache_2.adb:27
```

Contacter une tâche absente (disparue)

- Contacter une tâche absente
  - `Tasking_Error` propagée à l'appelant
  - L'appel ne se fait pas
    - Levée de l'exception hors de l'`accept`
    - Rattrapage possible mais pas dans l'`accept`

- Rappel utile...

  ... on peut tester l'état d'une tâche

  - `<tâche>'Callable`
  - `<tâche>'Terminated`

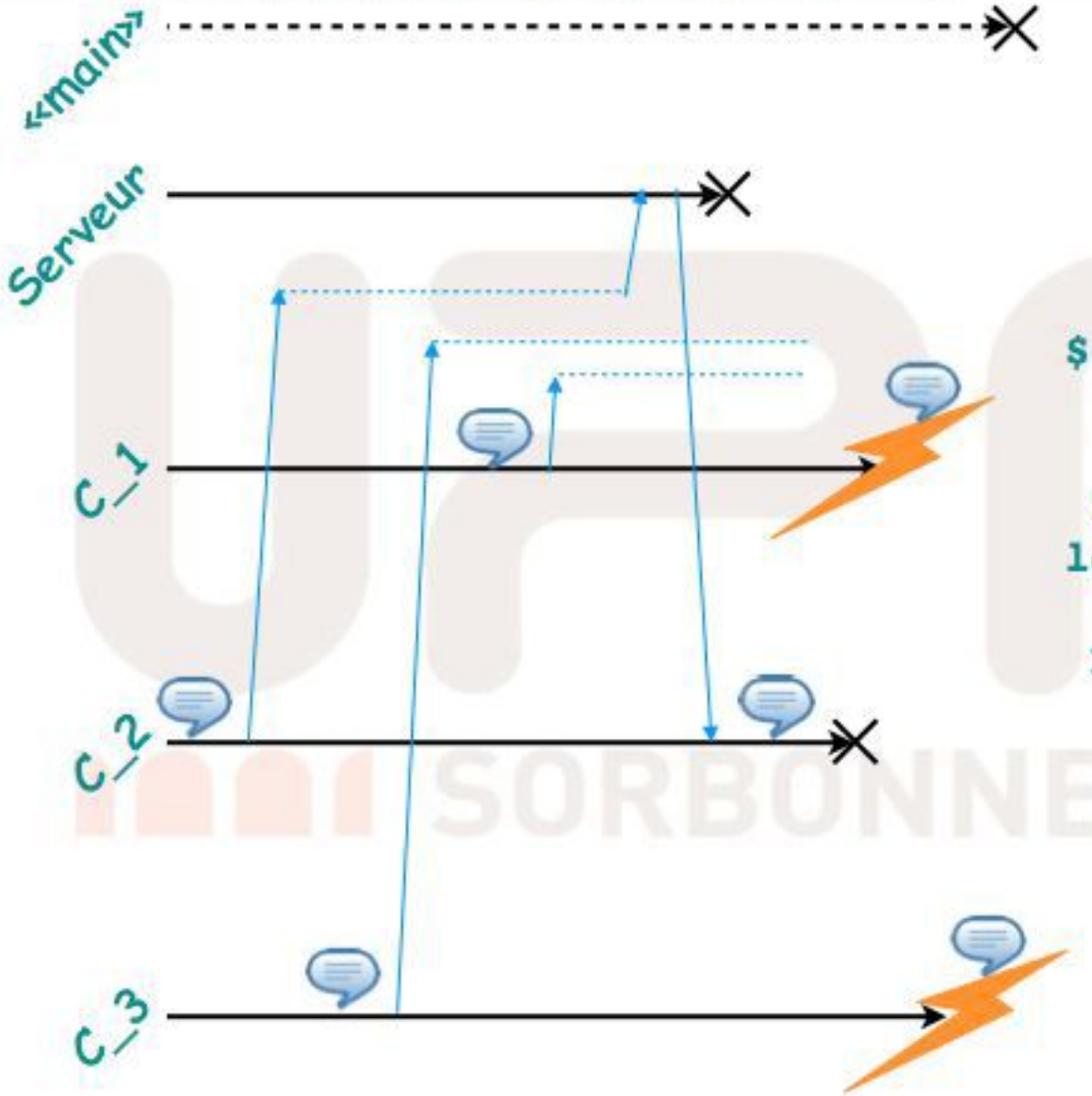
Exemple 3 : effet d'une disparition (1)

```
with Ada.Text_IO, Ada.Exceptions;
   use Ada.Text_IO, Ada;
   procedure Example_Exception_Tache_3 is
      task type Client (Id : Positive);
      task Serveur is
         entry Service (X : in Positive);
      end Serveur;
      task body Client is
      begin
         Put_Line ((1 .. Id => ' ') & "le Client" &
                   Positive'Image (Id) & " va appeler un service");
         Serveur.Service (Id);
         Put_Line ((1 .. Id => ' ') & "le Client" &
                   Positive'Image (Id) & " se termine");
      exception
         when E : others =>
            Put_Line ((1 .. Id => ' ') & "Le client" &
                      Positive'Image (Id) & " meurt sur " &
                      Exceptions.Exception_Name (E));
      end Client;
...
```

Exemple 3 : effet d'une disparition (2)


```
   task body Serveur is
   begin
      accept Service (X : in Positive) do
         Put_Line ("le serveur traite une demande de " &
                   Positive'Image (X));
      end Service;-- le serveur ne traite qu'un seul service
   end Serveur;
   C_1 : Client (1);
   C_2 : Client (2);
   C_3 : Client (3);
begin
   null;
end Example_Exception_Tache_3;
```

Le chronogramme correspondant




```
$ ./example_exception_tache_3
  le Client 2 va appeler un service
   le Client 3 va appeler un service
 le Client 1 va appeler un service
le serveur traite une demande de  2
  le Client 2 se termine
 Le client 1 meurt sur TASKING_ERROR
   Le client 3 meurt sur TASKING_ERROR
```

Exemple 4: étude de la terminaison (1)

- La mort d'une tâche n'entraîne pas la fin de l'application!
  - Confinement...

```
with Ada.Text_IO, Ada.Exceptions;
   use Ada.Text_IO, Ada;
   procedure Example_Exception_Tache_4 is
      task T1; -- Transmet une valeur à T2 via service
      task T2 is -- Attend une valeur qu'elle transmet à T3 via Service
         entry Service (V : in Integer);
      end T2;
      task T3 is -- Attend un appel
         entry Service (V : in Integer);
      end T3;
   ...
```

Exemple 4: étude de la terminaison (2)


```
...

   task body T1 is
   begin
      Put_Line ("T1 appelle un service de T2");
      T2.Service (-1);
      Put_Line ("T1 se termine");
   exception
      when E : others =>
         Put_Line ("T1 meurt sur " & Exceptions.Exception_Name
(E));
   end T1;

...
```

Exemple 4: étude de la terminaison (3)


```
...
   task body T2 is

         My_Value : Natural;

      begin
         Put_Line ("T2 Attend un nombre");
         accept Service (V : in Integer) do
            My_Value := V;
         end Service;
         Put_Line ("T2 appelle T3");
         T3.Service (My_Value);
         Put_Line ("T2 se termine");
      exception
         when E : others =>
            Put_Line ("T2 meurt sur " & Exceptions.Exception_Name
   (E));
      end T2;
```

Exemple 4: étude de la terminaison (4)

```
...
   task body T3 is
         My_Value : Natural;
      begin
         Put_Line ("T3 Attend un nombre");
         accept Service (V : in Integer) do
            My_Value := V;
         end Service;
         Put_Line ("T3 se termine avec My_Value =" &
                   Natural'Image (My_Value));
      exception
         when E : others =>
            Put_Line ("T3 meurt sur " & Exceptions.Exception_Name
   (E));
      end T3;
   begin
      null;
   end Example_Exception_Tache_4;
```

T3 ne se termine pas!

Le chronogramme correspondant