Slide 1

8 RPC et Rendez-vous

Communication par échange de messages = communication *uni*-directionnelle
(de l'envoyeur vers le récepteur)

Approche clients/serveurs ⇒ communication *bi*-directionnelle

- Un client envoie une requête à un serveur
- Le serveur traite la requête et retourne une réponse

Une mise en oeuvre de clients/serveurs avec échange de messages est possible :

⇒ Deux communications explicites distinctes

⇒ Nécessite deux canaux

Slide 2

Stratégie alternative : RPC et Rendez-vous

⇒ Communication bi-directionnelle

≈ appel de procédure permettant de recevoir des arguments *et* de retourner des résultats *en une seule opération*

Différences entre RPC et Rendez-vous

- RPC ⇒ chaque appel de procédure génère un processus indépendant (du coté de l'appelé) qui se synchronise avec le processus de l'appelant
- Rendez-vous ⇒ rencontre *synchrone* avec un processus unique (du coté de l'appelé) qui traite l'appel de procédure et qui se synchronise avec le processus de l'appelant

Slide 3

8.1 *Remote Procedure Call* **(Appel de procédure à distance)**

Utilise des *modules* ayant les caractéristiques suivantes :

- Une interface indique les *opérations publiques* (*exportées*)
- Une partie privée indique :
  - Des variables privées et du code d'initialisation
  - Des *procédures* qui réalisent les opérations publiques
  - Des procédures et/ou processus *privés* au module

Propriétés :

- Les appels aux opérations publiques peuvent être faits d'un autre processus (*remotely*, donc d'un espace mémoire distinct)
- Chaque appel génère un nouveau processus $\Rightarrow$ des mécanismes d'exclusion mutuelle doivent être utilisés pour accéder aux ressources du module
- L'appelant est suspendu jusqu'à ce que l'opération complète et que les résultats aient été reçus

Slide 4

Exemple : Un module pour gérer un compteur

```
module Compteur
  op incrementer( int i );
  op decrementer( int i );
  op valeur     ( int *res );
body
  int val = 0;
  sem mutex = 1;

  proc incrementer( int i )
  { P(mutex); val += i; V(mutex); }

  proc decrementer( int i )
  { P(mutex); val -= i; V(mutex); }

  proc valeur( int *res )
  { P(mutex); *res = val; V(mutex); }
end Compteur
```

Utilisation :

```
Compteur.incrementer(1);

Compteur.decrementer(2);

Compteur.val(&x);
```

Slide 5

8.2 Rendez-vous

Contrairement à l'approche RPC, l'approche par rendez-vous combine communication, synchronisation, *et* exclusion mutuelle

- Une requête pour une opération est traitée par un processus déjà existant associé au module
- Le serveur utilise une instruction spéciale de rendez-vous avec *gardes* et *sélection non-déterministe* $\Rightarrow$ une seule opération à la fois est traitée par le serveur $\Rightarrow$ exclusion mutuelle *implicite*
- L'appelant est bloqué jusqu'à ce que le rendez-vous soit complété, y compris le transfert des résultats
- Diffère d'un moniteur en ce que le module est un processus actif, alors qu'un moniteur est passif (le *thread* qui s'exécute dans un moniteur est celui de l'appelant)

Slide 6

Exemple : Un module pour gérer un compteur

```
module Compteur
  op incrementer( int i );
  op decrementer( int i );
  op valeur     ( int *res );
body
  process Compteur
  {
    int val = 0;

    while(true)
      in
        incrementer( i ) -> val += i;
        []
        decrementer( i ) -> val -= i;
        []
        valeur( res )    -> *res = val;
      ni
  }
end Compteur
```

Slide 7

8.5 Étude de cas : Java

Support pour RPC = RMI = *Remote Method Invocation*

```
interface RemoteCompteur extends Remote {
  void incrementer( int i ) throws RemoteException;
  void decrementer( int i ) throws RemoteException;
  int  valeur     ()        throws RemoteException;
}

class Client {
  public static void main(String[] args) {
    try {
      System.setSecurityManager( new RMISecurityManager() );
      String nomServeur = "rmi://arabica.info.uqam.ca:9996/compteur";
      RemoteCompteur rc = (RemoteCompteur) Naming.lookup(nomServeur);
      rc.incrementer(2);
      rc.decrementer(1);
      System.out.println( "Valeur lue = " + rc.valeur() );
    } catch (Exception e) { System.err.println(e); }
  }
}
```

Slide 8

```
class ServeurCompteur extends UnicastRemoteObject implements RemoteCompteur {
  protected int val = 0;

  public synchronized void incrementer( int i ) throws RemoteException
  { val += i; }

  public synchronized void decrementer( int i ) throws RemoteException
  { val -= i; }

  public synchronized int  valeur() throws RemoteException
  { return(val); }

  public ServeurCompteur() throws RemoteException
  { super(); }

  public static void main( String[] args ) {
    try {
      ServeurCompteur sc = new ServeurCompteur();
      String nomServeur = "rmi://arabica.info.uqam.ca:9996/compteur";
      Naming.bind(nomServeur, sc);
      System.out.println( "Le serveur est en fonction" );
    } catch (Exception e) { System.err.println(e); }
  }
}
```

Slide 9

8.6 Étude de cas : Ada

Ada a introduit et popularisé l'approche par rendez-vous :

- Ada 83 : Rendez-vous = unique mécanisme de programmation concurrente
- Ada 95 : Rendez-vous + Types protégés (*protected types* = moniteurs)

Exemple : Un module pour gérer un compteur

```
procedure CompteurProc
  task Compteur is
    entry incrementer( i:   IN  Natural );
    entry decrementer( i:   IN  Natural );
    entry valeur     ( res: OUT Integer );
  end;

  task body Compteur is separate;
end CompteurProc
```

Slide 10

```
task body Compteur is
  val: Integer;
begin
  val := 0;
  loop
    select
     accept incrementer( i: IN Natural )    do val := val + i; end;
     or
     accept decrementer( i: IN Natural )    do val := val - i; end;
     or
     accept valeur     ( res: OUT Integer ) do res := val;     end;
    end select;
  end loop
end Compteur;
```

Slide 11

Autre exemple : Un module pour gérer un compteur *non-négatif* . . .

- qui n'accepte des requêtes que si elles assurent que le compteur restera non-négatif
- qui termine si aucune requête n'a été reçue après 10 secondes

```
task body Compteur is
  val: Integer;
begin
  val := 0;
  loop
    select accept incrementer( i: IN Natural ) do
                  val := val + i;
           end;
    or     accept when val > 0 => decrementer( i: IN Natural ) do
                  val := val - i;
           end;
    or     accept valeur( res: OUT Natural ) do
                  res := val;
           end;
    or     delay 10.0; exit;
    end select;
  end loop
end Compteur;
```