SYSTÈMES ET RÉSEAUX INFORMATIQUES

COURS B4 : HTO(19339) et ICPJ(21937)
CYCLE PROBATOIRE INFORMATIQUE
(Conception et développement informatique)

CORRIGÉ DE L'EXAMEN DU 22 JUIN 2002

portant sur l'enseignement des SYSTÈMES INFORMATIQUES

CORRIGÉ PROBLÈME A. GESTION DES RESSOURCES

RÉPONSE ÀUX QUESTIONS A1. A1a = d ; A1b = c ; A1c = c ;

RÉPONSE À LA QUESTION A2.

La machine T (marque Tortue) respecte les échéances. En effet à t = 1, l'algorithme EDF préempte A pour élire B (intervalle 1..16). Puis quand B est fini, A est à nouveau élu (intervalle 16..285).

La machine L (marque Lièvre) ne respecte pas l'échéance de B. En effet il n'y a pas de préemption et A s'exécute pendant l'intervalle 0..27 avant que l'ordonnanceur n'élise B pendant l'intervalle 27..28,5. mais l'échéance de B (c'est 21) est dépassée.

Jean a intérêt à garder sa machine T ou bien à se programmer un ordonnanceur EDF pour L.

-- CORRIGÉ PROBLÈME B. SYNCHRONISATION DE PROCESSUS
-- LA FÊTE DES CAMÉNÉONS (programmes complets)

```
--                              ANNEXE  GÉNÉRALE
-- ************************          paquetage Semaphores          **
package Semaphores is
    type Semaphore is limited private;
    procedure P(S : in out Semaphore); procedure V(S: in out Semaphore);
    procedure E0(S : in out Semaphore; Val: in natural);
private
    protected type Sem is
        entry P;
        entry V;
        procedure E0(Val: Natural);
    private
        Compteur : Natural := 0;
        Init : Boolean := False;
    end Sem;
    type Semaphore is new Sem;
end Semaphores;
----------------------------------------
package body Semaphores is
    ----------------------------
    protected body Sem is
        entry P when (Compteur >0 ) and (Init) is
        begin
            Compteur := Compteur - 1;
```

```
        end P;
        entry V when Init is
        begin
            Compteur := Compteur + 1;
        end V;
        procedure E0(Val : in Natural) is
        begin
            Compteur := Val;
            Init := True;
        end E0;
    end Sem;
    ------------------------------------
    procedure P(S : in out Semaphore) is
    begin S.P; end P;

    procedure V(S : in out Semaphore) is
    begin S.V; end V;

    procedure E0(S : in out Semaphore; Val: in natural) is
    begin S.E0(Val); end E0;
    ---------------------------------------
end Semaphores;
-- ************************                    paquetage Carnation    **
package Carnation is
    type Couleur is (Bleu, Jaune, Rouge);
    NbCameneons : Constant Integer := 50;
    type IdProc is mod NbCameneons; -- type modulo, range 0..NbCameneons-1
    function GetCouleur(X : in IdProc) return Couleur;  -- c'est la couleur de X
    procedure Mutation(X, Y : in out IdProc) ;
end Carnation;

package body Carnation is
    -- table contenant la couleur de chaque caménéon
    LaCouleur : array(IdProc) of Couleur :=
                (IdProc'First => Couleur'First,
                IdProc'Last => Couleur'Last,
                others => Couleur'Succ(Couleur'First));
    -- le premier est Bleu, le dernier est Rouge, les autres sont Jaunes
    -- LaCouleur(IdProc'First) := Couleur'First; -- LaCouleur(1) = Bleu
    -- LaCouleur(IdProc'Last) := Couleur'Last; -- LaCouleur( Last) = Rouge
    -- le langage Ada permet de donner des attributs à un type numérique
    -- Couleur'First = Bleu; Couleur'Last = Rouge; Couleur'Succ(Bleu)= Jaune
    -- Couleur'Pos(Bleu) = 0; Couleur'Pos(Jaune) := 1; Couleur'Pos(Rouge) = 2
    -- Couleur'Val(0) = Bleu; Couleur'Val(1) = Jaune; Couleur'Val(2) = Rouge;

    procedure Mutation(X, Y : in out IdProc) is
    begin
        if LaCouleur(X) /= LaCouleur(Y) then
        LaCouleur(Y) := Couleur'Val(3 -
                                    Couleur'Pos(LaCouleur(X)) -
                                    Couleur'Pos(LaCouleur(Y)));
        LaCouleur(X) := LaCouleur(Y);
        end if;
    end Mutation;

    function GetCouleur(X : IdProc) return Couleur is
    begin return LaCouleur(X); end GetCouleur;
```

```
begin null; end Carnation;
--************************          paquetage Nom **
with Carnation; use Carnation;

package Nom is
      procedure Unique(X : out IdProc);
end Nom;

with Semaphores; use Semaphores;

package body Nom is
      Mutex : Semaphore;
      Valeur : IdProc := IdProc'First;
      procedure Unique(X : out IdProc) is
      begin
      P(Mutex);
      X := Valeur; Valeur := IdProc'Succ(Valeur);
      V(Mutex);
      end Unique;
begin E0(Mutex, 1); end Nom;
--****************************          paquetage Mail **
with Carnation; use Carnation;

package Mail is
      procedure Rencontre(X : in out IdProc) ; -- avec un autre caméneon
end Mail ;

-- ************************          procedure Main **

with Nom; with Mail; with Carnation; use Carnation;
with Ada.Text_IO; use Ada.Text_IO;

procedure Main is

      task type Animal;
      Seconds : constant Duration := 1.0;

      task body Animal is
            MaCouleur, Y : Couleur;
            Ego : IdProc;
      begin
            Nom.Unique(Ego); -- acquiert un numero unique
            MaCouleur := GetCouleur(Ego);
            for I in 1..50 loop
                  --  ce delai simule une vie en dehors du mail
                   delay Duration((Float(Ego)/Float(I))*Seconds);
                  Mail.Rencontre(Ego);
                  Y:= GetCouleur(Ego); -- pour voir si elle a changé
                  if Y /= MaCouleur then
                        Put_Line("Le caméneon" & IdProc'Image(Ego) &" : "&
                              Couleur'Image(MaCouleur)& "=>" & Couleur'Image(Y ));
                        MaCouleur := Y;
                  end if;
            end loop;
      end Animal;
      Cameneon : array(IdProc) of Animal;
begin null; end Main;
-- €€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€
```

-- ~~@@@@@@@~~ **ANNEXE MAIL UN (questions B1a et B1b)** ~~@@@@@@@~~

--- ********** processus magicien sorcier et sémaphores **********

-- ********** **paquetage Mail** **

```
with Semaphores; use Semaphores;

package body Mail is

-- *********                                   paquetage Requete **
   package Requete is
      procedure Deposer(X : in IdProc);
      procedure Retirer(Y : out IdProc);
   end Requete;

   package body Requete is
      Nplein, Nvide, Mutexprod : Semaphore;        -- <<1>> < question B1a>
      type Index is mod 5;
      T : array(Index) of IdProc;
      Tete , Queue : Index := 0;
      procedure Deposer(X : in IdProc) is
      begin                                        -- <<2>> <question B1a>
         P(Nvide);
         P(Mutexprod); T(Queue) := X; Queue := Queue + 1; V(Mutexprod);
         V(Nplein);
      end Deposer;
      procedure Retirer(Y : out IdProc) is
      begin                                        -- <<3>> <question B1a>
         P(Nplein);
         Y := T(Tete); Tete := Tete + 1;
         V(Nvide);
      end Retirer;
   begin
      E0(Nvide, 5); E0(Nplein, 0) ; E0(Mutexprod, 1);  -- <<9>> <question B1a>
   end Requete;

-- *****************                          déclaration des sémaphores ***

   Mutex: Semaphore;                            -- <<4>> <question B1b>
   Sem : array(IdProc) of Semaphore;

-- ******************                          procedure Rencontre **
   procedure Rencontre(X: in out IdProc) is
   begin
      Requete.Deposer(X);  -- dépose sa demande
      P(Sem(X));   -- attend le signal pour sortir de la procedure -- <<5>> <question B1a>
   end Rencontre;

-- *********************                       processus Magicien **
   task Magicien;
   task body Magicien is
      A, B : IdProc;
   begin
      loop
         Requete.Retirer(A);
         Requete.Retirer(B);
         P(Mutex) ;  -- pour la coherence quand il y a le sorcier -- <<6>> <question B1b>
```

```
                Mutation(A, B);
                V(Mutex); -- quand il y a le sorcier                    -- <<6>> <question B1b>
                V(Sem(A); V(Sem(B));                                    -- <<7>> <question B1a>
            end loop;
        end Magicien ;

-- *************************                              processus Sorcier          **
        task Sorcier;
        task body Sorcier is
            I, J : IdProc := IdProc’First;
        begin
            for K in 1..10*NbCameneons loop
                P(Mutex) ;  -- pour la coherence quand il y a le sorcier -- <<8>> <question B1b>
                Mutation(I, J);
                V(Mutex); -- quand il y a le sorcier                    -- <<8>> <question B1b>
                I := I + 2; J := J + 3; -- tout est : modulo NbCameneons
            end loop;
        end Sorcier;

--  *******************************

begin                              -- <<9>> <question B1a> et  <<10>> <question B1b>
        E0(Mutex, 1); for I in IdProc loop E0(Sem(I), 0); end loop;
end Mail;
--*****************************************************
-- €€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€
--                          ANNEXE  MAIL  DEUX  (question B2a)
--- ******************          sémaphores privés       ****************

with Semaphores; use Semaphores;

package body Mail is

--  *****************            déclaration des sémaphores et variables du paquetage Mail   ***
        Mutex: Semaphore;                                       -- <<1>> < question B2a>
        Sem : array(IdProc) of Semaphore;
        Premier : Boolean := True; -- variable persistante
        A, B : IdProc; -- variables persistantes
        -- Mutex assure la coherence des variables persistantes

--  ******************                                 procedure Rencontre       **
        procedure Rencontre(X: in out IdProc) is
        begin                                                   -- <<2>> < question B2a>
            P(Mutex);
            if Premier then
                A := X; Premier := False; -- pour le prochain appel
                -- ne pas oublier le blocage du processus appelant
            else
                B := X; Premier := True; -- pour le prochain appel
                Mutation(A, B);
                V(Sem(A)); V(Sem(B)); -- libere les deux processus
            end if;
            V(Mutex);
            P(Sem(X)); -- seul le premier processus peut être bloqué
        end Rencontre;
```

```
-- ******************          initialisations de tous les sémaphores  **
begin                                                  -- <<3>> < question B2a>
    E0(Mutex, 1); for I in IdProc loop E0(Sem(I), 0); end loop;
end Mail;

-- ***************************************************************************

--**************  variante de programmation des semaphores prives
-- <<2>> < question B2a>
--   procedure Rencontre(X: in out IdProc) is -- autre programmation possible
--       Attendre : Boolean; -- variable locale à l'appelant
--   begin
--       P(Mutex);
--       if Premier then
--           A := X; Premier := False; -- pour le prochain appel
--           Attendre := True; -- noter le blocage du processus appelant
--       else
--           B := X; Premier := True; -- pour le prochain appel
--           Mutation(A, B);
--           Attendre :=False; V(Sem(A)); -- reveille le premier qui attend
--       end if;
--       V(Mutex);
--       if Attendre then P(Sem(X)); end if; -- Sem(A) serait faux : A est partagé et modifiable
--       -- seul le premier processus peut être bloqué. Attendre et X sont locaux au processus
--   end Rencontre;

--**************  deuxième variante de programmation des semaphores prives
-- <<2>> < question B2a>
--   procedure Rencontre(X: in out IdProc) is -- autre programmation possible
--   begin
--       P(Mutex);
--       if Premier then
--           A := X; Premier := False; -- pour le prochain appel
--           V(Mutex);
--            P(Sem(X)); -- X est local. Sem(A) serait faux car A est partagé et modifiable
--       else
--           B := X; Premier := True; -- pour le prochain appel
--           Mutation(A, B);
--            V(Sem(A)); -- reveille le premier qui attend
--           V(Mutex);
--       end if;
--       V(Mutex);
--   end Rencontre;
*****************************************************************************
--ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
--                  ANNEXE  MAIL  DEUX  BIS  (question B2b)
-- ******************** variante avec section critique courte  **********
--- ******************       sémaphores privés       ****************

with Semaphores; use Semaphores;

package body Mail is
-- ******************          déclaration des sémaphores et variables du paquetage Mail ***
    Mutex: Semaphore;                          -- <<1>> < question B2b>
    Sem : array(IdProc) of Semaphore;
```

```
    Premier : Boolean := True ; -- variable persistante
    SauveA: IdProc; -- variable persistante
    -- Mutex assure la coherence des variables persistantes

-- ******************                    procedure Rencontre      **
    procedure Rencontre(X: in out IdProc) is
        A, B: IdProc; -- variables locales qui ne servent qu'au 2eme processus
    begin                                          -- <<2>> < question B2b>
        P(Mutex);
        if Premier then
            SauveA := X; Premier := False; -- pour le prochain appel
            V(Mutex);
            P(Sem(X));
        else
            A := SauveA; Premier := True; -- pour le prochain appel
            V(Mutex);
            B := X; -- ici A et B sont locaux et appartiennent au processus appelant
            Mutation(A, B);
            V(Sem(A)); -- libère l'autre processus une fois la mutation faite
        end if;
    end Rencontre;
-- ******************          initialisations de tous les sémaphores  **
begin                                              -- <<3>> < question B2b>
    E0(Mutex, 1); for I in IdProc loop E0(Sem(I), 0); end loop;
end Mail;

-- €€€€€€  RÉPONSE à la question B2c : Sbloc.E = 2 ou 1 (selon schéma choisi), Non.
-- €€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€
--            ANNEXE MAIL DEUX TER (hors examen)
--- ******************      sémaphores privés      ****************
-- ************* solution avec passage de droit d'exclusion mutuelle
--Le deuxième processus, B, ne rend pas le verrou d'acces en exclusion mutuelle,
-- mais il le passe à A, qui ne le rendra qu'apres avoir ete reveille.
-- Si A n'est elu que tardivement, tous les processus sont alors aussi retardes
-- On peut alors n'avoir qu'un semaphore Signal, de blocage
-- Cette methode de synchronisation est proche de celle des objets protégés de Ada

with Semaphores; use Semaphores;

package body Mail is
-- ******************          déclaration des sémaphores et variables du paquetage Mail  ***
    Mutex: Semaphore;
    Signal : Semaphore;
    Premier : Boolean := True;          -- variable persistante
    A, B : IdProc;                      -- variables persistantes
    -- Mutex assure la coherence des variables persistantes

-- ******************                    procedure Rencontre      **
    procedure Rencontre(X: in out IdProc) is
    begin
        P(Mutex);
        if Premier then
            A := X; Premier := False; -- pour le prochain appel
            V(Mutex); -- termine son passage en exclusion mutuelle
            --  blocage du premier d'une paire de processus appelant
            P(Signal); -- B lui passera le témoin d'exclusion mutuelle
            V(mutex); -- termine le passage de la paire en exclusion mutuelle
        else
```

```
            B := X; Premier := True; -- pour le prochain appel
            Mutation(A, B);
            V(Signal); -- réveille A chargé de rendre l'exclusion mutuelle
        end if;
    end Rencontre;

begin
    E0(Mutex, 1); E0(Signal, 0);
end Mail;
-- €€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€
--                ANNEXE   MAIL TROIS  : VERSION UN  (question B3)

--************** solution avec tache serveur en Ada avec rendez-vous *****
--   ** pour simplifier l'écriture des solutions, on n'utilise pas la clause terminate

--  ********************                         nouveau paquetage Nom   adb     **
package body Nom is
    task Serveur is
        entry Unique(X : out IdProc);
    end Serveur ;

    task body Serveur is
        Valeur : IdProc := IdProc'First;
    begin
        loop
            accept Unique(X : out IdProc) do
                X := Valeur; Valeur := IdProc'Succ(Valeur); end Unique;
        end loop;
    end Serveur ;

    procedure Unique(X : out IdProc) is
    begin  Serveur.Unique(X : out IdProc) ; end Unique;
end Nom;

-- ***************                                        paquetage Mail          **

package body Mail is                             -- solution avec tâche serveur Ada

-- *********                                           tache serveur Requete  **

    task  Requete is
        entry Deposer(X : in IdProc);
        entry Retirer(Y : out IdProc);
    end Requete;

    package body Requete is                      -- <<1>> < question B3>
        type Index is mod 5;
        T : array(Index) of IdProc;
        Tete , Queue : Index := 0; Nombre : Integer := 0;
    begin
        loop
            select
                when Nombre < 5 => accept Deposer(X : in IdProc) do
                    T(Queue) := X;
                end Deposer;
                Queue := Queue + 1; Nombre := Nombre + 1;
            or
                when Nombre > 0 => accept Retirer(Y : out IdProc) do
```

```
                    Y := T(Tete);
                end Retirer;
                Tete := Tete + 1; Nombre := Nombre - 1;
            end select;
        end loop;
    end Requete;

-- *********                                    tableau de taches serveur Signal  **
    task type UnSignal is
        entry Attendre;
        entry Envoyer;
    end UnSignal;

    task body UnSignal is
        SignalRecu : Boolean := False;
    begin                                                  -- <<2>> < question B3>
        loop
            select
                when SignalRecu => accept Attendre; SignalRecu := False;
            or
                accept Envoyer; SignalRecu := True;
            end select;
        end loop;
    end UnSignal;

    Signal : array(IdProc) of UnSignal;
-- ******************                                  procedure Rencontre      **
    procedure Rencontre(X: in out IdProc) is
    begin
        Requete.Deposer(X);  -- dépose sa demande
        Signal(X).Attendre;   -- attend le signal pour sortir de la procedure
    end Rencontre;
-- **********************                                  processus Magicien      **
    task Magicien;
    task body Magicien is
        A, B : IdProc;
    begin
        loop
            Requete.Retirer(A);
            Requete.Retirer(B);
            Mutation(A, B);
            Signal(A).Envoyer; Signal(B).Envoyer;
        end loop;
    end Magicien ;

begin null; end Mail;
-- ************************************************************************************
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
--                ANNEXE   MAIL   TROIS : VERSION DEUX
--                      (autre solution pour Question B3)
--********** solution avec tache serveur en Ada avec rendez-vous *****
--   ** pour simplifier l'écriture des solutions, on n'utilise pas la clause terminate
-- ***************                                  paquetage Mail          **

package body Mail is                        -- solution avec tâche serveur Ada

-- *****************                                  processus Magicien      **
    task Magicien is
        entry Rencontre1(X : in out IdProc);
```

```
        entry Rencontre2(Y : in out IdProc);
    end Magicien ;

    task body Magicien is
    begin
        loop                                          -- <<1>> < question B3>
            accept Rencontre1(X : in out IdProc) do
                accept Rencontre2(Y : in out IdProc) do
                    Mutation(X, Y);
                end Rencontre2;
            end Rencontre1;
        end loop;
    end Magicien ;
-- ****************************                       procedure Rencontre      **
    procedure Rencontre(X: in out IdProc) is
    begin
        if (X mod 2 = 0) then
            Magicien.Rencontre1(X);
        else
            Magicien.Rencontre2(X);
        end if;
    end Rencontre;

begin null; end Mail;
-- ********************************************************************************
--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  VARIANTE (hors examen)  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    task Placez is                    --  tache pour placer premier ou second un cameneon
        entry Moi(B : out Boolean);
    end Placez;
    task body Placez is
        Premier : Boolean := True;
    begin
        loop
            accept Moi(B : out Boolean) do B := Premier; end Moi;
            Premier := not Premier;
        end loop;                     -- lecture et modification sont faites en section critique
    end Placez;

    procedure Rencontre(X: in out IdProc) is
        Premier : Boolean;
    begin
        placez.Moi(Premier);
        if Premier then
            Magicien.Rencontre1(X);
        else
            Magicien.Rencontre2(X);
        end if;
    end Rencontre;
-- ********************************************************************************
--~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
--              ANNEXE   MAIL   TROIS : VERSION TROIS  (hors examen)
--                      (autre solution pour Question B3)
--********** solution avec tache serveur en Ada avec rendez-vous et requeue  *****
--   ** pour simplifier l'écriture des solutions, on n'utilise pas la clause terminate
-- ***************                                   paquetage Mail        **

package body Mail is                      -- solution avec tâche serveur Ada

-- *****************                                 processus Serveur     **
```

```
    task Serveur is
        entry Rencontre(X : in out IdProc);
        entry Attendre(X : in out IdProc);
    end Magicien ;

    task body Serveur is
        Premier, Ouvert : Boolean := True; -- variables persistantes
        Signal : Boolean := False;
        P, Q : IdProc;
        -- Ouvert sert a interdire un 3e accept de Rencontre avant l'accept de Attendre
    begin
        loop
            select
                when Ouvert => accept Rencontre(X : in out IdProc) do
                    if Premier then
                        P := X; Premier := False; -- pour le prochain appel
                        Signal := False;
                        requeue Attendre;
                    else
                        Q := X ; Premier := True; -- pour le prochain appel
                        Mutation(P, Q);
                        Signal := True; Ouvert := False;
                    end if
                end Rencontre;
            or
                when Signal => accept Attendre(X : in out IdProc) do
                    Signal := False; Ouvert := True;
                end Attendre;
            end select;
        end loop;
    end Serveur ;
-- ****************************                    procedure Rencontre      **
    procedure Rencontre(X: in out IdProc) is
    begin
        Serveur.Rencontre(X);
    end Rencontre;

begin null; end Mail;

-- ***************************************************************************
--~~~~~~~~~~~~~~~~~~~~~~~~~   ANNEXE MAIL QUATRE   ~~~~~~~~~~~~~~~~
--**********  solution avec objet protégé en Ada (hors examen)  *******
-- objet protégé avec requeue
-- ************************                    nouveau  paquetage Nom      **
package body Nom is
    protected Serveur is
        procedure Unique(X : out IdProc);
    private
        Valeur : IdProc := IdProc'First;
    end Serveur ;

    protected body Serveur is
        procedure Unique(X : out IdProc) is
        begin
            X := Valeur; Valeur := IdProc'Succ(Valeur);
        end Unique;
    end Serveur ;
```

```
      procedure Unique(X : out IdProc) is
      begin Serveur.Unique(X : out IdProc) ; end Unique;
end Nom;

-- **************                              paquetage Mail        **
package body Mail is                           -- solution avec objet protégé Ada

--******************                           objet protégé Magic    **
      protected Magic is
            entry Rencontre(X : in out IdProc) ;
      private
            Premier : Boolean := True; -- variables persistantes
            Signal : Boolean := False;
            P, Q : IdProc;
            entry Attendre(X : in out IdProc);
      end Magic ;

      protected body Magic is
            entry Rencontre(X : in out IdProc) when True is
            begin
                  if Premier then
                        P := X; Premier := False; -- pour le prochain appel
                        Signal := False;
                        requeue Attendre;
                  else
                        Q := X ; Premier := True; -- pour le prochain appel
                        Mutation(P, Q);
                        Signal := True;
                  end if;
            end Rencontre;
            entry Attendre(X : in out IdProc) when Signal is
            begin
                  Signal := False;
            end Attendre;
            -- La semantique de l'objet protege (schema de l'oeuf), donne priorité au reveil de processus
            -- le processus A bloque sur l'entree Attente dont la garde (Signal) est devenue "True"
            -- sera prioritaire par rapport a une nouvelle (et 3eme) l'execution de l'entree Rencontre
            -- On evite que ce 3e processus empeche le reveil de A en remettant Signal a "False"
      end Magic;

--  ******************************             procedure Rencontre     **
      procedure Rencontre(X: in out IdProc) is
      begin
            Magic.Rencontre(X);
      end Rencontre;
--  ***********
begin null; end Mail;

--€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€€
```