AMIARD Landry
CROQUEFER Gaëtan

2018/2019

Rapport TP sur les Processus Concurrents

UE. Paradigmes de programmation

I – Algorithmes de Dekker et Peterson

Dekker 1 :

```
processN:integer:=1;

task processus1;
task body processus1 is

I:integer:=0;

begin
  for I in 1..10 loop
    -- attente active:
    while (processN=2) loop
      Put_Line("Process1 en attente");
    end loop;
    -- <SC>
    Put_Line ("processus1 en SC");
    -- <SC>
    -- protocole de sortie:
    processN:=2;
  end loop;
end processus1;
```

```
task processus2;
task body processus2 is

J:integer:=0;

Begin
  for J in 1..10 loop
  -- attente active:
    while (processN=1) loop
      Put_Line ("processus2 en attente");
    end loop;
    -- <SC>
    Put_Line ("processus2 en SC");
    -- <SC>
    -- protocole de sortie:
    processN:=1;
  end loop;
end processus2;
```

Dekker 2 :

```
P1inside:boolean:=FALSE;
P2inside:boolean:=FALSE;

task processus1;
task body processus1 is

I:integer:=0;

begin
  for I in 1..10 loop
    -- attente active:
    while (P2inside) loop
      Put_Line("Process1 en attente");
    end loop;
    P1inside:=TRUE;
    -- <SC>
    Put_Line ("processus1 en SC");
    -- <SC>
    -- protocole de sortie:
    P1inside:=FALSE;
  end loop;
end processus1;
```

```
task processus2;
task body processus2 is

J:integer:=0;

Begin
  for J in 1..10 loop
  -- attente active:
  while (P1inside) loop
    Put_Line("Process2 en attente");
  end loop;
  P2inside:=TRUE;
    -- <SC>
  Put_Line ("processus2 en SC");
    -- <SC>
  -- protocole de sortie:
  P2inside:=FALSE;
  end loop;
end processus2;
```

Dekker 3 :

```
P1Wantstoenter:boolean:=FALSE;
P2Wantstoenter:boolean:=FALSE;

task processus1;
task body processus1 is

I:integer:=0;

begin
  for I in 1..10 loop
    -- protocole d'entree:
    P1Wantstoenter:=TRUE;
    -- attente active:
    while (P2Wantstoenter) loop
      Put_Line("Process1 en attente");
    end loop;
    -- <SC>
  Put_Line ("processus1 en SC");
    -- <SC>
  -- protocole de sortie:
  P1Wantstoenter:=FALSE;
  end loop;
end processus1;
```

```
task processus2;
task body processus2 is

J:integer:=0;

Begin
  for I in 1..10 loop
    -- protocole d'entree:
    P2Wantstoenter:=TRUE;
    -- attente active:
    while (P1Wantstoenter) loop
      Put_Line("Process2 en attente");
    end loop;
    -- <SC>
    Put_Line ("processus2 en SC");
    -- <SC>
    -- protocole de sortie:
    P2Wantstoenter:=FALSE;
  end loop;
end processus2;
```

- Dekker 4 étant quasi équivalent à Dekker 3, nous ne l'avons pas traité-

Peterson :

```
tour:integer:=1;
demande1:boolean:=FALSE;
demande2:boolean:=FALSE;

task processus1;
task body processus1 is

I:integer:=0;

begin
  for I in 1..2 loop
    -- protocole d'entree:
    demande1:=TRUE;
    tour:=2;
    --attente active:
    while (demande2=TRUE AND tour/=1) loop
      Put_Line("Process1 en attente");
    end loop;
    -- <SC>
  Put_Line ("processus1 en SC");
    -- <SC>
  -- protocole de sortie:
  demande1:=FALSE;
  end loop;
end processus1;
```

```
task processus2;
task body processus2 is

J:integer:=0;

Begin
  for I in 1..2 loop
    -- protocole d'entree:
    demande2:=TRUE;
    -- attente active:
    tour:=1;
    while (demande1=TRUE AND tour/=2) loop
      Put_Line("Process2 en attente");
    end loop;
    -- <SC>
    Put_Line ("processus2 en SC");
    -- <SC>
    -- protocole de sortie:
    demande2:=FALSE;

  end loop;
end processus2;
```

Peterson symétrique :

```
procedure Peter_Sym is
package int_io is new Integer_io(integer);
use int_io;

Demande: array(0..1) of Boolean := (others => FALSE);
Tour:Integer:=0;
J:Integer:=0;

procedure Entre(I: in Integer) is
begin
  J:=(I+1) mod 2;
  Demande(i):=TRUE;
  Tour:=J;
  while (Demande(J)=True and Tour /= 1) loop null;
end loop;
end Entre;

procedure Sortie(I: in Integer) is
begin
  Demande(i):=FALSE;
end Sortie;


task Process1;
task body Process1 is

  begin
    for I in 0..5 loop
      Entre(0);
      -- <SC>
      Put_Line("Process1 en Section Critique");
      -- <SC>
      Sortie(0);
    end loop;
  end Process1;

  task Process2;
  task body Process2 is

    begin
      for I in 0..5 loop
        Entre(1);
        -- <SC>
        Put_Line("Process2 en Section Critique");
        -- <SC>
        Sortie(1);
      end loop;
    end Process2;

  begin
    Null;
  end Peter_Sym;
```

II - Problème des cheminots

Premier cas :

Pierre=FAUX ;

```
Processus Péruvien
Début
   Tant que VRAI faire
   Début
          /*Peruvien arrête train*/
(a1)      Tant que (Pierre=VRAI) faire
          Début
             /*Péruvien fait la sieste*/
          Fin
          /*Péruvien cherche et mets la pierre*/
(b1)      Pierre=VRAI ;
          <SC>
             /*Péruvien passe*/
          </SC>
          /*Péruvien retourne au panier et enlève la pierre*/
(c1)      Pierre=FAUX ;
          /*Péruvien repart*/
   Fin
Fin
```

```
Processus Bolivien
Début
   Tant que VRAI faire
   Début
          /*Bolivien arrête train*/
(a2)      Tant que (Pierre=VRAI) faire
          Début
             /*Bolivien fait la sieste*/
          Fin
          /*Bolivien cherche et mets la pierre*/
(b2)      Pierre=VRAI ;
          <SC>
             /*Bolivien passe*/
          </SC>
          /*Bolivien retourne au panier et enlève la pierre*/
(c2)      Pierre=FAUX ;
          /*Bolivien repart*/
   Fin
Fin
```

Propriété 1 :

Pierre : FAUX

Péruvien	Bolivien	Pierre
(a1)		
	(a2)	
(b1)		VRAI
	(b2)	VRAI
<SC>		
	<SC>	

PP1 NON VÉRIFIÉE

Propriété 2 :

Si Péruvien demande la <SC> et est le seul à la demander, il peut y entrer car Pierre=FAUX : il n'y a donc pas d'attente.
Idem pour Bolivien.
PP2 VÉRIFIÉE

Propriété 3 :

Supposons Péruvien en <SC> et Bolivien en attente :
=>Pierre=VRAI
Péruvien sort de <SC> et éxécute son protocole de sortie : Pierre=FAUX. Bolivien peut maintenant, sortir de son attente active en (a2).
PP3 VÉRIFIÉE

Interblocage :

Pierre est initialisé à FAUX, et est affecté à FAUX lors du protocole de sortie des deux processus : il ne peut jamais être VRAI lorsque les deux processus avancent dans leur instruction (a1) et (a2).
PAS D'INTERBLOCAGE

Equité :

Supposons Péruvien en <SC> et Bolivien en attente :
=>Pierre=VRAI
Péruvien sort de <SC> et éxécute son protocole de sortie : Pierre=FAUX.
Péruvien passe en protocole d'entrée puis redemande la <SC> : Il l'obtient.
NON EQUITABLE

Dans ce premier cas, la propriété 1 n'est pas respecté ; si les deux processus peuvent entrer en section critique simultanément, les deux trains peuvent entrer en collision… Il n'y a pas de priorité donné à qui que ce soit, c'est pour cela qu'il y a un accident.

Deuxième cas :

Pierre=FAUX ;

```
Processus Péruvien
Début
   Tant que VRAI faire
   Début
         /*Peruvien arrête train*/
(a1)     Tant que (Pierre=FAUX) faire
         Début
            /*Péruvien fait la sieste*/
         Fin
         <SC>
            /*Péruvien passe*/
         </SC>
         /*Péruvien retourne au panier et enlève la pierre*/
(b1)     Pierre=FAUX ;
         /*Péruvien repart*/
   Fin
Fin
```

```
Processus Bolivien
Début
   Tant que VRAI faire
   Début
         /*Bolivien arrête train*/
(a2)     Tant que (Pierre=VRAI) faire
         Début
            /*Bolivien fait la sieste*/
         Fin
         <SC>
            /*Bolivien passe*/
         </SC>
         /*Bolivien retourne au panier et met une pierre*/
(b2)     Pierre=VRAI;
         /*Bolivien repart*/
   Fin
Fin
```

Propriété 1 :

Supposons Péruvien et Bolivien en <SC>
• Péruvien en <SC> => Pierre=VRAI
• Bolivien en <SC> => Pierre=FAUX
CONTRADICTION, PP1 VÉRIFIÉE

Propriété 2 :

Après initialisation, si Bolivien demande la <SC> et est le seul à la demander, il peut y entrer car Pierre=FAUX : il n'y a donc pas d'attente.
En revanche, si Péruvien demande la <SC> et est le seul à la demander, il ne peut pas y entrer. Il est obligé d'attendre que Péruvien mette Pierre à VRAI.
PP2 NON VÉRIFIÉE

Propriété 3 :

Supposons Péruvien en <SC> et Bolivien en attente :
=> Pierre=VRAI
Péruvien sort de <SC> et éxécute son protocole de sortie : Pierre=FAUX. Bolivien peut maintenant sortir de son attente active en (a2).
PP3 VÉRIFIÉE

Interblocage :

La variable Pierre est booléenne, elle ne peut prendre que VRAI ou FAUX :
- Si Pierre=VRAI alors Péruvien passe.
- Si Pierre=FAUX alors Bolivien passe.
PAS D'INTERBLOCAGE

Equité :

Supposons Péruvien en <SC> et Bolivien en attente :
=>Pierre=VRAI
Péruvien sort de <SC> et éxécute son protocole de sortie : Pierre=FAUX.
Péruvien passe en protocole d'entrée puis redemande la <SC> : Il ne l'obtient pas, Bolivien entre en <SC>.
ÉQUITABLE

Dans ce deuxième cas, la propriété 2 n'est pas respecté ; si un processus est prêt à entrer en section critique, il ne peut pas y acceder sans attendre l'autre. Dans la situation des cheminots, c'est pour cette raison qu'il n'y a pas autant de trains qu'avant.. La condition d'attente est trop contraignante, il faut la modifier si on veut obtenir un acces à la sectionc ritique plus facile.

Troisième cas :

Pierre1=FAUX, Pierre2=FAUX ;

```
Processus Péruvien
Début
   Tant que VRAI faire
   Début
            /*Peruvien arrête train et met sa pierre*/
(a1)        Pierre1=VRAI ;
(b1)        Tant que (Pierre2=VRAI) faire
            Début
(c1)           Pierre1=FAUX ;
               /*Péruvien enlève sa pierre et fais la sieste*/
            Fin
            /*Peruvien remet sa pierre */
            Pierre1=VRAI
            <SC>
               /*Péruvien passe*/
            </SC>
            /*Péruvien retourne au panier et enlève sa pierre*/
(c1)        Pierre1=FAUX ;
            /*Péruvien repart*/
   Fin
Fin
```

```
Processus Bolivien
Début
   Tant que VRAI faire
   Début
            /*Bolivien arrête train et met sa pierre*/
(a1)        Pierre2=VRAI ;
(b2)        Tant que (Pierre1=VRAI) faire
            Début
               /*Bolivien enlève sa pierre et fais la sieste*/
(c2)           Pierre2=FAUX ;
            Fin
            /*Bolivien remet sa pierre*/
            Pierre2=VRAI ;
            <SC>
               /*Bolivien passe*/
            </SC>
            /*Bolivien retourne au panier et enlève sa pierre*/
(c2)        Pierre=FAUX ;
            /*Bolivien repart*/
   Fin
Fin
```

Propriété 1 :

Supposons Péruvien en <SC> :
=>Pierre1=VRAI
=>Pierre2=FAUX
Supposons Bolivien en <SC> :
=>Pierre2=VRAI
=>Pierre1=FAUX
CONTRADICTION, PP1 VÉRIFIÉE

Propriété 2 :

Après initialisation, si Péruvien demande la <SC> et est le seul à la demander, il peut y entrer car Pierre2=FAUX : il n'y a donc pas d'attente.
De même si Bolivien demande la <SC> et est le seul à demander, il y entre : Pierre1=FAUX.
PP2 VÉRIFIÉE

Propriété 3 :

Supposons Péruvien en <SC> et Bolivien en attente :
=> Pierre1=VRAI
=> Pierre2=FAUX
Péruvien sort de <SC> et éxécute son protocole de sortie : Pierre1=FAUX. Bolivien peut maintenant sortir de son attente active en (b2) et entrer en <SC>.
PP3 VÉRIFIÉE

Interblocage :

Les variables Pierre1 et Pierre2 sont booléennes, elle ne peuvent prendre comme valeur que VRAI ou FAUX. Le changement de valeur qui permet de sortir de la condition d'attente se trouve dans le processus opposé à celui qui attend : dès lors que l'un des deux processus change la valeur pour l'autre il débloque ce dernier et l'envoie en <SC>.
PAS D'INTERBLOCAGE

Equité :

Supposons Péruvien en <SC> et Bolivien en attente :
=> Pierre1=VRAI
=> Pierre2=FAUX
Péruvien sort de <SC> et éxécute son protocole de sortie : Pierre1=FAUX.
Péruvien repasse en protocole d'entrée puis redemande la <SC> : Il l'obtient, en passant devant le Bolivien.
NON ÉQUITABLE

Solution proposé :

La solution au problème consiste à implementer l'algorithme de Peterson pour les cheminots :

Pierre1=FAUX ; Pierre 2=FAUX ; tour=1

```
Processus Péruvien
Début
   Tant que VRAI faire
   Début
         /*Peruvien arrête train et met sa pierre*/
(a1)     Pierre1=VRAI ;
         /*Peruvien donne le tour à Bolivien*/
(b1)     tour=2 ;
(c1)     Tant que (Pierre2=VRAI ET tour=2) faire
         Début
            /*Péruvien enlève sa pierre et fais la sieste*/
         Fin
         <SC>
            /*Péruvien passe*/
         </SC>
         /*Péruvien retourne au panier et enlève sa pierre*/
(d1)     Pierre1=FAUX ;
         /*Péruvien repart*/
   Fin
Fin
```

```
Processus Bolivien
Début
   Tant que VRAI faire
   Début
         /*Bolivien arrête train et met sa pierre*/
(a2)     Pierre2=VRAI ;
         /*Bolivien donne le tour à Péruvien*/
(b2)     tour=1 ;
(c2)     Tant que (Pierre1=VRAI ET tour=1) faire
         Début
            /*Bolivien enlève sa pierre et fais la sieste*/
         Fin
         <SC>
            /*Bolivien passe*/
         </SC>
         /*Bolivien retourne au panier et enlève sa pierre*/
(d2)     Pierre2=FAUX ;
         /*Bolivien repart*/
   Fin
Fin
```

Propriété 1 :

Supposons Péruvien en <SC> :
=>Pierre1=VRAI
=>Pierre2=FAUX
=>tour=2
Supposons Bolivien en <SC> :
=>Pierre2=VRAI
=>Pierre1=FAUX
=>tour=1
CONTRADICTION, PP1 VÉRIFIÉE

Propriété 2 :

Après initialisation, si Péruvien demande la <SC> et est le seul à la demander, il peut y entrer car Pierre2=FAUX : il n'y a donc pas d'attente.
De même si Bolivien demande la <SC> et est le seul à demander, il y entre : Pierre1=FAUX.
PP2 VÉRIFIÉE

Propriété 3 :

Supposons Péruvien en <SC> et Bolivien en attente :
=> Pierre1=VRAI
=> Pierre2=FAUX
=> tour=2
Péruvien sort de <SC> et éxécute son protocole de sortie : Pierre1=FAUX. Bolivien peut maintenant sortir de son attente active en (c2) et entrer en <SC>
PP3 VÉRIFIÉE

Interblocage :

La variable tour est booléenne : il est impossible pour elle d'avoir deux valeurs ne même temps. Sachant qu'une parti nécessaire de la condition d'attente en découle, il est impossible d'avoir les deux processus en attente.
PAS D'INTERBLOCAGE

Equité :

La variable tour sert à donner la priorité à un processus. Si c'est le tour du processus 2 alors celui-ci est prioritaire à la section critique si le processus 1 n'y est pas déjà. Un processus peut toutefois s'éxecuter plusieurs fois si l'autre ne souhaite pas acceder à la section critique.
ÉQUITABLE

Dans ce dernier cas, l'ensemble des propriétés est respecté : on peut appliquer cet manière de faire à notre problème ; cela devrait régler les problèmes féroviaires dans les Andes.

Les codes ADA de ces questions sont disponibles en annexe, nous avons jugé plus facile de démontrer les propriétés sous forme de pseudo-code (c'est aussi plus agréable à l'oeil).

III- Généralisation à 3 processus

Généralisation brutale :

tour=1 ; demande1=FAUX ; demande2=FAUX ; demande3=FAUX

```
Processus 1
Début
   Tant que VRAI faire
   Début
(a1)      demande1=VRAI ;
(b1)      tour=2 ;
(c1)      Tant que (tour!=1) ET (demande2=VRAI
          OU demande3=VRAI) faire
          Début
             /*Attente*/
          Fin
          <SC></SC>
(d1)      demande1=FAUX ;
   Fin
Fin
```

```
Processus 2
Début
   Tant que VRAI faire
   Début
(a2)      demande2=VRAI ;
(b2)      tour=3 ;
(c2)      Tant que (tour!=2) ET (demande1=VRAI
          OU demande3=VRAI) faire
          Début
             /*Attente*/
          Fin
          <SC></SC>
(d2)      demande2=FAUX ;
   Fin
Fin
```

```
Processus 3
Début
   Tant que VRAI faire
   Début
(a3)      demande3=VRAI ;
(b3)      tour=1 ;
(c3)      Tant que (tour!=3) ET (demande1=VRAI
          OU demande2=VRAI) faire
          Début
             /*Attente*/
          Fin
          <SC></SC>
(d3)      demande3=FAUX ;
   Fin
Fin
```

Propriété 1 :

Processus 1	Processus 2	Processus 3	demande1	demande2	demande3	tour
(a1)			VRAI			
		(a3)			VRAI	
(b1)						2
		(b3)				1
<SC>						
	(a2)			VRAI		
	(b2)					3
		<SC>				

PP1 NON VÉRIFIÉE

Propriété 2 :

Après initialisation, le processus 1 demande la section critique et est le seul à la demander, il peut y entrer car les deux autres ne la demande pas: il n'y a donc pas d'attente.
Idem pour le processus 2 et le processus 3.
PP2 VÉRIFIÉE

Propriété 3 :

La propriété 2 n'étant pas vérifié, le seul moyen qu'a un processus de rentrer en section critique lorsqu'il est en attente dépend donc du protocole de sortie du programme en <SC>. Dans le cas présent, tous les processus donnent la main dans le protocole de sortie : ce qui débloque un autre processus en attente.
PP3 VÉRIFIÉE

Interblocage :

On suppose les trois processus en attente :
process1 en attente => tour!=1
process2 en attente => tour!=2
process1 en attente => tour!=3
Les valeurs des autres sont négligeables. Ici, tour devrait prendre une quatrième valeur pour satisfaire les conditions, or, tour ne peut prendre que trois valeur ; il y a donc forcément une condition qui n'est pas satisfaite.
PAS D'INTERBLOCAGE

Équité :

Avant chaque passage en section critique, un processus est obligé de passer la main au suivant via la variable tour. Il est impossible donc d'avoir le même processus s'exécuter en boucle.
ÉQUITABLE

Le code ADA de cette version est également disponible en annexe

Solution proposé pour 3 processus concurrents :

Pour trouver une solution au problème des 3 processus, il faut identifier ce qui pose problème dans la solution globalisé de Peterson. Le principale soucis viens du fait qu'il est possible sous certaines conditions d'avoir simultanément plusieurs processus en section critique. Il faut donc un moyen de contrôle sur qui peut prendre la section critique lorsque plusieurs processus la demande en même temps.

Nous avons trouvé un moyen arbitraire de le faire, notre solution vérifie la propriété 1. En revanche, elle n'est plus très équitable, le processus 1 est prioritaire par rapport aux deux autres.

```
Processus 1

Début
   Tant que VRAI faire
   Début
(a1)    demande1=VRAI;
(b1)    Si (demande2=VRAI) et
        (demande3=FAUX)
           tour=2 ;
(c1)       Sinon Si(demande2=FAUX) et
           (demande3=VRAI)
              tour=3 ;
           Fin Si
        Fin Si
(d1)    Si (demande2=VRAI) et
        (demande3=VRAI)
           tour=1 ;
        Fin Si
(e1)    Tant que ((tour!=1) ET (demande2=VRAI
        OU demande3=VRAI)) faire
        Début
              /*Attente*/
        Fin
        <SC></SC>
(f1)    demande1=FAUX ;
    Fin
Fin
```

```
Processus 2

Début
   Tant que VRAI faire
   Début
(a2)    demande2=VRAI;
(b2)    Si (demande1=VRAI) et
        (demande3=FAUX)
           tour=1 ;
(c2)       Sinon Si(demande1=FAUX) et
           (demande3=VRAI)
              tour=3 ;
           Fin Si
        Fin Si
(d2)    Si (demande1=VRAI) et
        (demande3=VRAI)
           tour=1 ;
        Fin Si
(e2)    Tant que ((tour!=2) ET (demande1=VRAI
        OU demande3=VRAI)) faire
        Début
              /*Attente*/
        Fin
        <SC></SC>
(f2)    demande3=FAUX ;
    Fin
Fin
```

```
Processus 3

Début
   Tant que VRAI faire
   Début
(a3)    demande3=VRAI;
(b3)    Si (demande1=VRAI) et
        (demande2=FAUX)
           tour=1 ;
(c3)       Sinon Si(demande1=FAUX) et
           (demande2=VRAI)
              tour=2 ;
           Fin Si
        Fin Si
(d3)    Si (demande1=VRAI) et
        (demande2=VRAI)
           tour=1 ;
        Fin Si
(e3)    Tant que ((tour!=3) ET (demande1=VRAI
        OU demande3=2RAI)) faire
        Début
              /*Attente*/
        Fin
        <SC></SC>
(f3)    demande2=FAUX ;
    Fin
Fin
```

Le code ADA de cette version se trouve en annexe

IV-Annexe

Cheminot 1 ADA :

```
pierre:boolean:=FALSE;

task perou;
task body perou is

I:integer:=0;


begin
  for I in 1..10 loop
    -- protocole d entree
    Put_Line("Peruvien arrete train");
    --attente active
    while(pierre=TRUE) loop
      Put_Line("Peruvien fait la sieste");
    end loop;
    Put_Line("Peruvien cherche pierre");
    Put_Line("Peruvien met pierre dans panier");
    pierre:=TRUE;
    -- <SC>
    Put_Line("Peruvien passe");
    -- <SC>
    -- protocole de sortie
    Put_Line("Peruvien retourne au panier");
    Put_Line("Peruvien enleve la pierre");
    pierre:=FALSE;
    Put_Line("Peruvien repart");
  end loop;
end perou;
```

```
task bolivie;
task body bolivie is

J:integer:=0;

begin
  for J in 1..10 loop
    -- protocole d entree
    Put_Line("Bolivien arrete train");
    -- attente active
    while(pierre=TRUE) loop
      Put_Line("Bolivien fait la sieste");
    end loop;
    Put_Line("Bolivien cherche pierre");
    Put_Line("Bolivien met pierre dans panier");
    pierre:=TRUE;
    -- <SC>
    Put_Line("Bolivien passe");
    -- <SC>
    -- protocole de sortie
    Put_Line("Bolivien retourne au panier");
    Put_Line("Bolivien enleve la pierre");
    pierre:=FALSE;
    Put_Line("Bolivien repart");

  end loop;
end bolivie;
```

Cheminot 2 ADA :

```
pierre:boolean:=FALSE;

task perou;
task body perou is

I:integer:=0;

begin
  for I in 1..10 loop
    -- protocole d entree
    Put_Line("Peruvien arrete train");
    -- attente active
    while(pierre=FALSE) loop
      Put_Line("Peruvien fait la sieste");
    end loop;
    -- <SC>
    Put_Line("Peruvien passe");
    -- <SC>
    -- protocole de sortie
    Put_Line("Peruvien retourne au panier");
    Put_Line("Peruvien enleve la pierre");
    pierre:=FALSE;
    Put_Line("Peruvien repart");
  end loop;
end perou;
```

```
task bolivie;
task body bolivie is

J:integer:=0;

begin
  for J in 1..10 loop
    -- protocole d entree
    Put_Line("Bolivien arrete train");
    -- attente active
    while(pierre=TRUE) loop
      Put_Line("Bolivien fait la sieste");
    end loop;
    -- <SC>
    Put_Line("Bolivien passe");
    -- <SC>
    -- protocole de sortie
    Put_Line("Bolivien retourne au panier");
    Put_Line("Bolivien met une pierre dans le panier");
    pierre:=TRUE;
    Put_Line("Bolivien repart");
  end loop;
end bolivie;
```

Cheminot 3 ADA :

```
pierre1:boolean:=FALSE;
pierre2:boolean:=FALSE;

task perou;
task body perou is

I:integer:=0;

begin
  for J in 1..10 loop
    -- protocole d entree
    Put_Line("Peruvien arrete son train et met sa pierre");
    pierre1:=TRUE;
    --attente active
    while(pierre2=TRUE) loop
      Put_Line("Peruvien enleve sa pierre et fait la sieste");
      pierre1:=FALSE;
    end loop;
    Put_Line("Peruvien remet sa pierre");
    pierre1:=true;
    -- <SC>
    Put_Line("Peruvien passe");
    -- <SC>
    -- protocole de sortie
    Put_Line("Peruvien retourne au panier");
    Put_Line("Peruvien enleve sa pierre");
    pierre1:=FALSE;
    Put_Line("Peruvien repart");
  end loop;
end perou;
```

```
task bolivie;
task body bolivie is

J:integer:=0;

begin
  for I in 1..10 loop
    -- protocole d entree
    Put_Line("Bolivien arrete train et met sa pierre");
    pierre2:=TRUE;
    --attente active
    while(pierre1=TRUE) loop
      Put_Line("Bolivien enleve sa pierre et fait la sieste");
      pierre2:=FALSE;
    end loop;
    Put_Line("Bolivien remet sa pierre")
    pierre1:=true;
    -- <SC>
    Put_Line("Bolivien passe");
    -- <SC>
    -- protocole de sortie
    Put_Line("Bolivien retourne au panier");
    Put_Line("Bolivien enleve sa pierre");
    pierre2:=FALSE;
    Put_Line("Bolivien repart");
  end loop;
end bolivie;
```

Solution cheminots ADA :

```
pierre1:boolean:=FALSE;
pierre2:boolean:=FALSE;
tour:integer:=1;

task perou;
task body perou is

I:integer:=0;

begin
  for J in 1..10 loop
    -- protocole d entree
    Put_Line("Peruvien arrete son train et met sa pierre");
    pierre1:=TRUE;
    Put_Line("Péruvien donne le tour au Bolivien");
    tour:=2;
    --attente active
    while(pierre2=TRUE and tour=2) loop
      Put_Line("Peruvien fait la sieste");
    end loop;
    -- <SC>
    Put_Line("Peruvien passe");
    -- <SC>
    -- protocole de sortie
    Put_Line("Peruvien retourne au panier");
    Put_Line("Peruvien enleve sa pierre");
    pierre1:=FALSE;
    Put_Line("Peruvien repart");
  end loop;
end perou;
```

```
task bolivie;
task body bolivie is

J:integer:=0;

begin

  for I in 1..10 loop
    -- protocole d entree
    Put_Line("Bolivien arrete son train et met sa pierre");
    pierre2:=TRUE;
    Put_Line("Bolivien donne le tour au Peruvien");
    tour:=1;

    --attente active
    while(pierre1=TRUE and tour=1) loop
      Put_Line("Bolivien fait la sieste");
    end loop;

    -- <SC>
    Put_Line("Bolivien passe");
    -- <SC>

    -- protocole de sortie
    Put_Line("Bolivien retourne au panier");
    Put_Line("Bolivien enleve sa pierre");
    pierre2:=FALSE;
    Put_Line("Bolivien repart");
  end loop;
end bolivie;
```

Peterson symétrique :

```
procedure Peterson_Sym is
package int_io is new Integer_io(integer);
use int_io;


Demande: array(0..1) of Boolean := (others => FALSE);
Tour:Integer:=0;
J:Integer:=0;

procedure Entre(I: in Integer) is
begin
  J:=(I+1) mod 2;
  Demande(i):=TRUE;
  Tour:=J;
  while (Demande(J)=True and Tour /= 1) loop null;
end loop;
end Entre;

procedure Sortie(I: in Integer) is
begin
  Demande(i):=FALSE;
end Sortie;


task Process1;
task body Process1 is

  begin
    for I in 0..5 loop
      Entre(0);
      Put_Line("Process1 en Section Critique");
      Sortie(0);
    end loop;
  end Process1;

 task Process2;
 task body Process2 is

    begin
      for I in 0.05 loop
        Entre(1);
        Put_Line("Process2 en Section Critique");
        Sortie(1);
      end loop;
    end Process2;

  begin
    Null;
  end Peterson_Sym;
```

Généralisation brutale Peterson :

```
tour:integer:=1;
demande1:boolean:=FALSE;
demande2:boolean:=FALSE;
demande3:boolean:=FALSE;

task processus1;
task body processus1 is

I:integer:=0;

begin
  for I in 1..2 loop
    -- protocole d'entree:
    demande1:=TRUE;
    tour:=2;
    --attente active:
    while (tour/=1) and (demande2=TRUE or demande3=TRUE) loop
      Put_Line("Process1 en attente");
    end loop;
    -- <SC>
  Put_Line ("processus1 en SC");
    -- <SC>
  -- protocole de sortie:
  demande1:=FALSE;
  end loop;
end processus1;
```

```
task processus2;
task body processus2 is

J:integer:=0;

Begin
  for J in 1..2 loop
    -- protocole d'entree:
    demande2:=TRUE;
    tour:=3;
    -- attente active:
    while (tour/=2) and (demande1=TRUE or demande3=TRUE) loop
      Put_Line("Process2 en attente");
    end loop;
    -- <SC>
    Put_Line ("processus2 en SC");
    -- <SC>
    -- protocole de sortie:
    demande2:=FALSE;

  end loop;
  d processus2;
```

```
task processus3;
task body processus3 is

K:integer:=0;

Begin
  for K in 1..2 loop
    -- protocole d'entree:
    demande2:=TRUE;
    tour:=1;
    -- attente active:
    while (tour/=2) and (demande1=TRUE or demande2=TRUE) loop
      Put_Line("Process3 en attente");
    end loop;
    -- <SC>
    Put_Line ("processus3 en SC");
    -- <SC>
    -- protocole de sortie:
    demande3:=FALSE;

  end loop;
end processus3;
```

Solution 3 processus :

```
tour:integer:=1;
demande1:boolean:=FALSE;
demande2:boolean:=FALSE;
demande3:boolean:=FALSE;

task processus1;
task body processus1 is

I:integer:=0;

begin
  for I in 1..2 loop
    -- protocole d'entree:
    demande1:=TRUE;
    if(demande2=TRUE and demande3=FALSE) then
        tour:=2;
    else if (demande3=TRUE and demande2=TRUE) then
        tour:=3;
        end if;
    end if;
    if(demande2=TRUE and demande3=TRUE)then
        tour:=1;
    end if;
    --attente active:
    while (tour/=1) and (demande2=TRUE or demande3=TRUE) loop
      Put_Line("Process1 en attente");
    end loop;
    -- <SC>
  Put_Line ("Process1 en SC");
    -- <SC>
  -- protocole de sortie:
  demande1:=FALSE;
  end loop;
end processus1;
```

```
task processus2;
task body processus2 is

J:integer:=0;

begin
  for J in 1..2 loop
    -- protocole d'entree:
    demande2:=TRUE;
    if(demande3=TRUE and demande1=FALSE) then
        tour:=3;
    else if (demande1=TRUE and demande3=FALSE) then
        tour:=1;
        end if;
    end if;
    if(demande3=TRUE and demande1=TRUE) then
        tour:=1;
    end if;
    --attente active:
    while (tour/=2) and (demande1=TRUE or demande3=TRUE) loop
      Put_Line("Process2 en attente");
    end loop;
    -- <SC>
  Put_Line ("Process2 en SC");
    -- <SC>
  -- protocole de sortie:
  demande2:=FALSE;
  end loop;
end processus2;
```

```
task processus3;
task body processus3 is

K:integer:=0;

begin
  for K in 1..2 loop
    -- protocole d'entree:
    demande3:=TRUE;
    if(demande1=TRUE and demande2=FALSE) then
        tour:=1;
    else if (demande2=TRUE and demande1=FALSE) then
        tour:=2;
        end if;
    end if;
    if(demande1=TRUE and demande2=TRUE) then
        tour:=1;
    end if;
    --attente active:
    while (tour/=3) and (demande1=TRUE or demande2=TRUE) loop
      Put_Line("Process3 en attente");
    end loop;
    -- <SC>
  Put_Line ("Process3 en SC");
    -- <SC>
  -- protocole de sortie:
  demande3:=FALSE;
  end loop;
end processus3;
```