Rapport de TP : Systèmes concurrents

Charles Follet, Roland Bary

Fichiers sources disponibles sur `https://github.com/cfollet/ada_concurr`

Master 1 Technologies de l'Internet
Université de Pau et des Pays de l'Adour

Sommaire

Introduction

Première partie
Rendez-vous ADA

L'objectif ici est de résoudre à l'aide de l'outil RDV ADA les problèmes de :
- Producteur/Consommateur
  - 1 Producteur/ 1Consommateur : tampon de taille N
  - 1 producteur/1 Consommateur avec un tampon de taille 1
- Lecteurs/Rédacteurs
  - Priorité aux Lecteurs
  - Priorité aux Rédacteurs
  - Priorité égales

1 Producteur/ Consommateur

1.1 1 Producteur/ 1 Consommateur : tampon de taille N

```
with TEXT_IO; use TEXT_IO;

--sauvegarder ce fichier sous le nom de: ProdCons.adb, et pour compiler lancer: gnatmake ProdCons.adb -o Prodcons


-- un producteur et un consommateur avec un tampon de taille N

procedure ProdCons is
    package int_io is new Integer_io(integer);
    use int_io;

    -- Interface  Magasinier
    task type Magasinier is
        entry produire(Mess : IN Integer);
        entry consommer(Mess : OUT Integer);
    end Magasinier;

    M : Magasinier;

    -- Interface Producteur
    task type Producteur is end Producteur;

    -- Interface consommateur
    task type Consommateur is end Consommateur;

    -- Body Magasinier
    task body Magasinier is
        cpt : Integer := 0;
        -- taille du tampon
        n : Integer := 8;
        tampon : array(0..n-1) of Integer;
        tete,queue : Integer range 0..n-1 := 0;
    begin
        loop
            select
                -- si le nombre de production est inférieur à la taille du tampon, on peut produire
                when (cpt < n) => accept produire(Mess : IN Integer)
                do
                    tampon(tete) := Mess;
                end;
                put_line("produire");
                tete := (tete + 1) mod n;
                cpt := cpt + 1;
            or
                -- s'il y a au moins une production, on peut consommer
                when (cpt > 0) => accept consommer(Mess : OUT Integer)
                do
```

```
                Mess := tampon(queue);
            end;
            put_line("consommer");
            queue := (queue + 1) mod n;
            cpt := cpt -1;
        end select;
    end loop;
end Magasinier;


-- corps Producteur
task body Producteur is
    value : Integer := 1;
begin
    for i in 1..10 loop
        M.produire(value);
    end loop;
end Producteur;

-- corps Consommateur
task body Consommateur is
    value : Integer := 0;
begin
    for i in 1..10 loop
        M.consommer(value);
    end loop;
end Consommateur;

p : Producteur;
c : Consommateur;

begin
    null;
end ProdCons;
```

1.2 Producteur/ Consommateur avec un tampon de taille 1

```
with TEXT_IO; use TEXT_IO;

-- Producteur consommateur avec un tampon de taille 1 :
    -- echange synchrone du message : le producteur ne peut produire que quand le lecteur a lu et inversement.

procedure ProdCons is
    package int_io is new Integer_io(integer);
    use int_io;
    cpt : Integer := 0;
    tampon : Integer;

    -- Interface Producteur
    task type Producteur is

    end Producteur;

    -- Interface consommateur
    task type Consommateur is
        entry echange(Mess : IN OUT Integer);
    end Consommateur;

    -- corps Consommateur
    task body Consommateur is
        Mess : Integer := 0;
    begin
        for i in 1..10 loop
            loop
                select
                    -- si le producteur a rempli le tampon
                    when (true) => accept echange(Mess : IN OUT Integer)
                    do
                        put_line("consommer");
                        Mess := tampon;
                        cpt := 0;
                    end;
                end select;
            end loop;
        end loop;
    end Consommateur;

    c : Consommateur;



    -- corps Producteur
    task body Producteur is
        Mess : Integer := 1;
    begin
        for i in 1..10 loop
        put_line("produire");
        tampon := Mess;
        cpt := 1;
        c.echange(tampon);
        end loop;
    end Producteur;


    p : Producteur;


begin
    null;
end ProdCons;
```

2 Lecteurs/ Rédacteurs

2.1 Priorité aux Lecteurs

```
with TEXT_IO; use TEXT_IO;

procedure LecteurRedacteur is
    package int_io is new Integer_io(integer);
    use int_io;

    -- Interface  Ecrivain
    task type Ecrivain is
        entry debut_lect;
        entry debut_red;
        entry fin_lect;
        entry fin_red;
    end Ecrivain;

    E : Ecrivain;

    task type Lecteur is end Lecteur;     -- Interface Lecteur
    task type Redacteur is end Redacteur; -- Interface Redacteur

    -- Body Ecrivain
    task body Ecrivain is
        nbLect : Integer := 0;
        nbRed : Integer := 0;
    begin
        loop
            select
                when (nbRed = 0) => accept debut_lect
                    do
                        put_line("debut_lect");
                        nbLect := nbLect + 1;
                    end;
            or
                when (nbLect+nbRed+debut_lect'count = 0) => accept debut_red
                    do
                    put_line("debut_red");
                    nbRed := nbRed + 1;
                    end;
            or
                accept fin_lect
                do
                    put_line("fin_lect");
                    nbLect := nbLect - 1;
                end;
            or
                accept fin_red
                do
                    put_line("fin_red");
                    nbRed := nbRed - 1;
                end;

            end select;
        end loop;
    end Ecrivain;

    task body Lecteur is
    begin
        for i in 1..10 loop
            E.debut_lect;
            E.fin_lect;
        end loop;
    end Lecteur;

    task body Redacteur is
    begin
        for i in 1..10 loop
            E.debut_red;
            E.fin_red;
        end loop;
    end Redacteur;
```

```
    l : Lecteur;
    r : Redacteur;

begin -- LecteurRedacteur
    null;
end LecteurRedacteur;
```

2.2 Priorité aux Rédacteurs

```
with TEXT_IO; use TEXT_IO;

procedure LecteurRedacteur is
    package int_io is new Integer_io(integer);
    use int_io;

    -- Interface  Ecrivain
    task type Ecrivain is
        entry debut_lect;
        entry debut_red;
        entry fin_lect;
        entry fin_red;
    end Ecrivain;

    E : Ecrivain;

    task type Lecteur is end Lecteur;     -- Interface Lecteur
    task type Redacteur is end Redacteur; -- Interface Redacteur

    -- Body Ecrivain
    task body Ecrivain is
        nbLect : Integer := 0;
        nbRed : Integer := 0;
    begin
        loop
            select
                when (nbRed+debut_red'count = 0) => accept debut_lect
                    do
                        put_line("debut_lect");
                        nbLect := nbLect + 1;
                    end;
            or
                when (nbLect+nbRed = 0) => accept debut_red
                    do
                    put_line("debut_red");
                    nbRed := nbRed + 1;
                    end;
            or
                accept fin_lect
                do
                    put_line("fin_lect");
                    nbLect := nbLect - 1;
                end;
            or
                accept fin_red
                do
                    put_line("fin_red");
                    nbRed := nbRed - 1;
                end;

            end select;
        end loop;
    end Ecrivain;

    task body Lecteur is
    begin
        for i in 1..10 loop
            E.debut_lect;
            E.fin_lect;
        end loop;
    end Lecteur;

    task body Redacteur is
    begin
        for i in 1..10 loop
            E.debut_red;
            E.fin_red;
        end loop;
    end Redacteur;

    l : Lecteur;
    r : Redacteur;
```

```
begin -- LecteurRedacteur
    null;
end LecteurRedacteur;
```

2.3 Priorités égales

2.3.1 Principe de la solution

Une solution naïve au problème aurait été de directement utiliser la solution avec les compteurs de Robert vu en cours, c'est à dire :

Lecture #act(ECRIRE) = 0

Ecriture #act(ECRIRE) + #act(LIRE) = 0

Si on utilise ces conditions, le cas suivant ne fonctionne pas :

L1 est en cours
    R1 arrive
    L2 arrive

En effet, rien n'empêche L2 de passer en lecture au détriment de R1.
Pour résoudre ce problème il faut que R1 signale son intention d'entrer en SC et nous avons choisis d'utiliser un drapeau.
Un lecteur/redacteur désirant aller en SC doit lever le drapeau(drapeau := 1). Ainsi, un lecteur/redacteur voulant entrer doit disposer de ce drapeau et le problème précédent est résolu.

2.3.2 Code ADA

```
with TEXT_IO; use TEXT_IO;

procedure LecteurRedacteur is
    package int_io is new Integer_io(integer);
    use int_io;

    -- Interface  Ecrivain
    task type Ecrivain is
        entry barriere;
        entry debut_lect;
        entry debut_red;
        entry fin_lect;
        entry fin_red;
    end Ecrivain;

    E : Ecrivain;

    task type Lecteur is end Lecteur;     -- Interface Lecteur
    task type Redacteur is end Redacteur; -- Interface Redacteur

    -- Body Ecrivain
    task body Ecrivain is
        nbLect : Integer := 0;
        nbRed : Integer := 0;
        drapeau : Integer := 0;
    begin
        loop
            select
                when (drapeau = 0) => accept barriere
                    do
                        drapeau := 1;
                    end;
            or
                when (nbRed = 0) => accept debut_lect
                    do
                        put_line("debut_lect");
                        nbLect := nbLect + 1;
                        drapeau := 0;
                    end;
            or
                when (nbLect+nbRed = 0) => accept debut_red
                    do
                        put_line("debut_red");
                        nbRed := nbRed + 1;
                    end;
```

```
            or
                accept fin_lect
                do
                    put_line("fin_lect");
                    nbLect := nbLect - 1;
                end;
            or
                accept fin_red
                do
                    put_line("fin_red");
                    nbRed := nbRed - 1;
                    drapeau := 0;
                end;

            end select;
        end loop;
    end Ecrivain;

    task body Lecteur is
    begin
        for i in 1..10 loop
            E.barriere;
            E.debut_lect;
            E.fin_lect;
        end loop;
    end Lecteur;

    task body Redacteur is
    begin
        for i in 1..10 loop
            E.barriere;
            E.debut_red;
            E.fin_red;
        end loop;
    end Redacteur;

    l : Lecteur;
    r : Redacteur;

begin -- LecteurRedacteur
    null;
end LecteurRedacteur;
```

2.3.3 Tests

Test 1 :

L1 est en cours

R1 arrive

L2 arrive

Actions	**nbLect**	**nbRed**	**drapeau**
L1.LIRE	1	0	0
arrivée de R1			
R1.barriere	1	0	1
R1.debutRed	R1 bloqué a debutRed		
arrivée de L2			
L2.barriere	L2 bloqué a barriere		
L1.finLect	0	0	1
R1.debutRed	0	1	1
R1.finRed	0	0	0
L2.debutLect	1	0	0
L2.finLect	0	0	0

Test 2 :

R1 est en cours

L1 arrive

R2 arrive

Actions	**nbLect**	**nbRed**	**drapeau**
R1.ecrire	0	1	1
arrivée de L1			
L1.barriere	L1 bloqué a barriere (FIFO)		
arrivée de R2			
R2.barriere	R2 bloqué a barriere (FIFO)		
R1.finRed	0	0	0
L1.debutLect	1	0	0
R2.debutRed	R2 bloqué a debutRed		
L1.finLect	0	0	0
R2.debutRed	0	1	1
R2.finRed	0	0	0

Deuxième partie

Objets/Types protégés ADA 95

L'objectif de cette partie est de :

— Rappeler le principe des Objets/Types protégés en ADA 95 : définition, principe, sémantique, différence avec les packages ADA

— Comparer les Objets/types protégés a d'autres outils comme les sémaphores, RDV ADA, : pouvoir d'expression, difficulté/facilité d'utilisation, etc..

— Implémenter à l'aide Objets/Types protégés :

- Producteur/Consommateur
- Lecteurs/ Rédacteurs : sans priorité et priorité aux lecteurs
- Le problème du Carrefour a sens giratoire

3 Rappel des principes des Objets/Types protégés en ADA 95

3.1 Définition

Un objet protégé définit des données privées ne pouvant être accédées que par les sous-programmes (fonctions, procédures ou entrées) associés à l'objet protégé.

3.2 Principe

Les objets protégés permettent l'encapsulation de données privées qui sont accédées par des fonctions (lecture des données), par des procédures (lectures/écritures des données semblables à celles d'un moniteur) et par des entrées (comme une procédure mais avec une mise en attente possible de l'appelant de la procédure) **La syntaxe est la suivante**

```
protected Nom_deL_Objet_Protege is
déclarations des fonctions, procédures et entrées de l'objet protégé
private
déclaration des données privées e l'objet protégé
end Nom_De_L_Objet_Protege;
```

3.3 Sémantique

3.3.1 Sémantique de base

— La partie privée définit les données ou contenu de l'objet protégé.

— Les fonctions définissent des actions de lecture du contenu de l'objet protégé (interdiction de modifier la valeur de celles-ci)

— Les procédures et les entrées définissent des actions de lecture et d'écriture des données de l'objet protégé (contrairement à une procédure ou à une fonction, l'appel à une entrée peut être bloquant)

— Plusieurs lectures peuvent avoir lieu simultanément

— Une action d'écriture exclut toute autre action (lecture ou écriture)

3.3.2 Sémantique des entrées

- Une entrée permet de définir des traitements sous conditions
  — Chaque entrée possède une file d'attente à laquelle est associée une expression booléenne que l'on appelle *garde* ou *barrière*
  — L'entrée n'est "ouverte" que si cette expression (la garde) est vraie
  — Lorsque l'entrée est *fermée* (la garde est fausse), les appels sur cette entrée sont mis en attente (ils ne seront traités que lorsque la garde redeviendra vraie)

- Une tâche, exécutant le code d'une entrée, peut être placée dans la file d'attente d'une autre entrée par l'instruction *requeue*
- Une tâche en attente sur une entrée interne est plus prioritaire qu'une tâche faisant un nouvel appel à une entrée (ou procédure) de l'objet protégé.

3.4 Différence avec les packages ADA

Il n'existe pas de différence particulière entre les Objets/Types protégés et les packages ADA. Ces deux notions ont toutes les deux une spécification (avec une partie privée) et un corps.

4 Implémentation à l'aide des Objets/Types protégés

4.1 Producteur/ Consommateur

```
with TEXT_IO; use TEXT_IO;

-- 1 producteur, 1 consommateur, tampon de taille n

procedure ProdConsObjetProtege is

    package int_io is new Integer_io(integer);
    use int_io;

    n:Integer:=8;
    type Tampon_Type is array (0..n-1) of Integer;

    -- interface Magasinier

    protected type Magasinier is
        entry produire(Mess : IN Integer);
        entry consommer(Mess : OUT Integer);
    private
            cpt : Integer := 0; --Compteur de remplissage du tampon
            tampon : tampon_Type;
            tete,queue : Integer range 0..n-1:= 0; --Tete et queue varient entre 0 et n-1
    end Magasinier;

    M : Magasinier;

    -- Interface Producteur
    task type Producteur is end Producteur;

    -- Interface Consommateur
    task type Consommateur is end Consommateur;

    --body  Magasinier
    protected body Magasinier is
        entry produire(Mess : IN Integer)
            when (cpt < n) is
        begin
            tampon(tete) := Mess;
            tete := (tete + 1) mod n;
            put_line("produire");
            cpt := cpt + 1;
        end produire;

        entry consommer(Mess : OUT Integer)
            when (cpt > 0) is
        begin
            Mess := tampon(queue);
            queue := (queue + 1) mod n;
            put_line("consommer");
            cpt := cpt-1;
        end consommer;

    end Magasinier;

    task body Producteur is
        value : Integer := 3;
    begin
        for i in 1..10 loop
            M.produire(value);
        end loop;
    end Producteur;


    task body Consommateur is
        value : Integer := 0;
    begin
        for i in 1..10 loop
            M.consommer(value);
        end loop;
    end Consommateur;
```

```
    P : Producteur;
    C : Consommateur;

    begin
        null;
end ProdConsObjetProtege;
```

4.2 Lecteurs/ Rédacteurs : sans priorité et priorité lecteurs en ADA 95

4.2.1 Priorité égale

```
with TEXT_IO; use TEXT_IO;

procedure LecteurRedacteur is

    package int_io is new Integer_io(integer);
    use int_io;

    -- interface Ecrivain

    protected type Ecrivain is
        entry debut_lect;
        entry debut_red;
        entry fin_lect;
        entry fin_red;
    private
        nbLect : Integer := 0;
        nbRed : Integer := 0;
    end Ecrivain;

    E : Ecrivain;

    task type Lecteur is end Lecteur;     -- Interface Lecteur
    task type Redacteur is end Redacteur; -- Interface Redacteur

    --body  Ecrivain
    protected body Ecrivain is
        entry debut_lect
            when (nbRed + debut_red'count = 0) is
        begin
            put_line("debut_lect");
            nbLect := nbLect + 1;
        end debut_lect;

        entry fin_lect
            when(true) is
        begin
            put_line("fin_lect");
            nbLect := nbLect - 1;
        end fin_lect;

        entry debut_red
            when (nbLect + debut_lect'count + nbRed = 0) is
        begin
            put_line("debut_red");
            nbRed := nbRed + 1;
        end debut_red;

        entry fin_red
            when(true) is
        begin
            put_line("fin_red");
            nbRed := nbRed - 1;
        end fin_red;
    end Ecrivain;

    task body Lecteur is
        value : Integer := 3;
    begin
        for i in 1..20 loop
            E.debut_lect;
            E.fin_lect;
        end loop;
    end Lecteur;


    task body Redacteur is
        value : Integer := 0;
    begin
        for i in 1..10 loop
            E.debut_red;
            E.fin_red;
```

```
        end loop;
    end Redacteur;

    L : Lecteur;
    R : Redacteur;

    begin
        null;
end LecteurRedacteur;
```

4.2.2 Priorité lecteurs

```
with TEXT_IO; use TEXT_IO;

procedure LecteurRedacteur is

    package int_io is new Integer_io(integer);
    use int_io;

    -- interface Ecrivain

    protected type Ecrivain is
        entry debut_lect;
        entry debut_red;
        entry fin_lect;
        entry fin_red;
    private
        nbLect : Integer := 0;
        nbRed : Integer := 0;
    end Ecrivain;

    E : Ecrivain;

    task type Lecteur is end Lecteur;     -- Interface Lecteur
    task type Redacteur is end Redacteur; -- Interface Redacteur

    --body  Ecrivain
    protected body Ecrivain is
        entry debut_lect
            when (nbRed = 0) is
        begin
            put_line("Lecture");
            nbLect := nbLect + 1;
        end debut_lect;

        entry fin_lect
            when(true) is
        begin
            put_line("fin_lect");
            nbLect := nbLect - 1;
        end fin_lect;

        entry debut_red
            when (nbLect+nbRed+debut_lect'count = 0) is
        begin
            put_line("debut_red");
            nbRed := nbRed + 1;
        end debut_red;

        entry fin_red
            when(true) is
        begin
            put_line("fin_red");
            nbRed := nbRed - 1;
        end fin_red;
    end Ecrivain;

    task body Lecteur is
        value : Integer := 3;
    begin
        for i in 1..10 loop
            E.debut_lect;
            put_line("Lecteur");
            E.fin_lect;
        end loop;
    end Lecteur;


    task body Redacteur is
        value : Integer := 0;
    begin
        for i in 1..10 loop
            E.debut_red;
            put_line("Redacteur");
            E.fin_red;
```

```
        end loop;
    end Redacteur;

    L : Lecteur;
    R : Redacteur;

    begin
        null;
end LecteurRedacteur;
```

4.3 Le problème du carrefour à sens giratoire

```
with TEXT_IO; use TEXT_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
procedure Giratoire is
    package int_io is new Integer_io(integer);
    use int_io;

    -- Declaration de variables
    capacite : Integer := 5;   -- Capacite du carrefour
    subtype type_voie is Integer range 0..10;

    -- Interface carrefour
    protected type Carrefour is
        -- Signal d'entree dans le carrefour
        entry entrer(type_voie);
        -- Signal de sortie du carrefour
        procedure sortir;

    private
        compteur : Integer := 0; -- Nombre de véhicule dans le carrefour
        carrefour_vide : Boolean := true;
        voie_courrante : type_voie;
    end Carrefour;

    protected body Carrefour is
        entry entrer(for numVoie in type_voie)
            when (numVoie = voie_courrante and compteur < capacite ) or (compteur=0 and carrefour_vide) is
        begin
            compteur := compteur +1 ;
            voie_courrante := numVoie;
            put_line("Voie courante :"  & Integer'Image(voie_courrante));
        end entrer;

        procedure sortir is
        begin
            compteur := compteur -1 ;
            -- Si on est le dernier vehicule,
            -- quil y ait ou non dautres véhicules qui attendent
            -- on passe la voie courante a la voie suivante et on dit que le carrefour est vide
            -- ainsi, si la voie suivante contient des vehicules ils passent,
            -- si elle nen contient pas, les autres peuvent passer aussi etant donné que
            -- compteur = 0 et carrefour_vide = true
            if compteur = 0
                then    voie_courrante := (voie_courrante+1) mod type_voie'Last;
                        carrefour_vide := true;
            end if;
        end sortir;
    end Carrefour;

    C : Carrefour;

    task type Voiture is end Voiture;
    task body Voiture is
    begin
        for i in type_voie loop
            C.entrer(i);
            put_line(Integer'Image(i));
            C.sortir;
        end loop;
    end Voiture;

v1 : Voiture;
v2 : Voiture;
v3 : Voiture;
v4 : Voiture;
v5 : Voiture;
v6 : Voiture;
v7 : Voiture;
v8 : Voiture;
v9 : Voiture;
v10 : Voiture;


begin -- Giratoire
```

```
    null;
end Giratoire;
```