Applications Concurrentes :

Conception,

Outils de Validation

ACCOV_B

Chapitre 4

CONCURRENCE AVEC DES TÂCHES ADA

invocation à distance
tâche serveur

DÉFINITIONS

PROCESSUS (ou TÂCHE en ADA) : unité de concurrence

Un programme concurrent commence séquentiellement avec une tâche principale ("main") qui crée (déclare) les autres processus concurrents.
Chacune des tâches déclarées se déroule en parallèle comme si il y avait autant de processeurs que de tâches. Chaque tâche utilise une pile.

L'APPEL DE PROCÉDURE (sous-programme : procédure ou fonction)

• une procédure donnée peut-être appelée de plusieurs endroits du programme (au départ un programme est une procédure principale "main")
• un appel de procédure entraîne un passage d'information de l'appelant vers l'environnement de la procédure appelée (in) ou réciproquement (out) ou dans les deux sens (in out)
• une procédure contient une série d'instructions (le corps de procédure) qui sont exécutées pour le compte de l'appelant, jusqu'au retour de procédure accompagné du passage de résultat : on dit que la procédure appelante "attend" le retour de la procédure appelée
(même si appelante et appelée sont exécutées pour le même processus)

APPEL DE PROCÉDURE DANS UN PROGRAMME CONCURRENT

• une procédure peut être appelée par un ou plusieurs processus
mais ce n'est pas une structure de communication ou synchronisation
• chaque appel crée un environnement indépendant (variables de travail et paramètres effectifs) qui ne communique pas avec les environnements des autres appels; les paramètres effectifs et les variables de travail sont rangés dans la pile du processus appelant et sont donc des données propres au processus (non partageables)
• Le code d'une procédure partagée doit rester invariant
(on dit que la procédure doit être réentrante)

L'INVOCATION À DISTANCE

• demande l'exécution d'une procédure à un autre processus
• est accompagnée d'un passage de messages entre processus
• comme pour l'appel de procédure, l'appelant "attend" la réponse de l'appelé, mais ici appelant et appelé ne sont pas le même processus
• son écriture ressemble à un appel de procédure
• les procédures invocables à distance sont des entrées du processus
• l'appel désigne le processus concerné et l'entrée appelée

L'APPEL D'UN OBJET PROTÉGÉ

Utilisation d'un objet de synchronisation partagé entre processus

LE RENDEZ-VOUS ENTRE TÂCHES DANS ADA 95

♦ rendez-vous asymétrique entre client appelant et serveur appelé

• le serveur est connu des clients;
mais le serveur ne connait pas les clients qui l'appellent

♦ coté serveur appelé

• points d'appel de la tâche appelée : déclaration d'entrée par entry

• acceptation d'un rendez-vous par l'appelé : instruction accept

• rendez-vous étendu : appel + traitement + retour

• passage de paramètres données in et résultats out

• acceptation gardée par une condition chez l'appelé :
when condition => accept
la condition ne peut contenir ni le nom ni de paramètre de l'appel

• fin du rendez-vous à l'initiative de l'appelé : accept ... do ... end; le!";'!entraine le retour des résultats puis le réveil de l'appelant
do...end permet au serveur de faire des actions client bloqué

• le rendez-vous peut être traité par plusieurs entrées successivement (deux!ou plus) : instruction requeue
sans libérer l'appelant au moment du requeue

• attente sélective (1 parmi P choix possibles) : clause select
unicité de l'accept : un seul rendez-vous quel que soit le nombre d'appels consultés par select
indéterminisme : rendez-vous avec l'un quelconque des possibles

• attente sélective avec choix de terminaison clause terminate
avec attente limitée clause delay [until]
avec rendez-vous conditionnel clause else

• notion de famille d'entrées

♦ coté client appelant

• appel procédural avec : nom_tâche.nom_entrée(paramètres_effectifs)

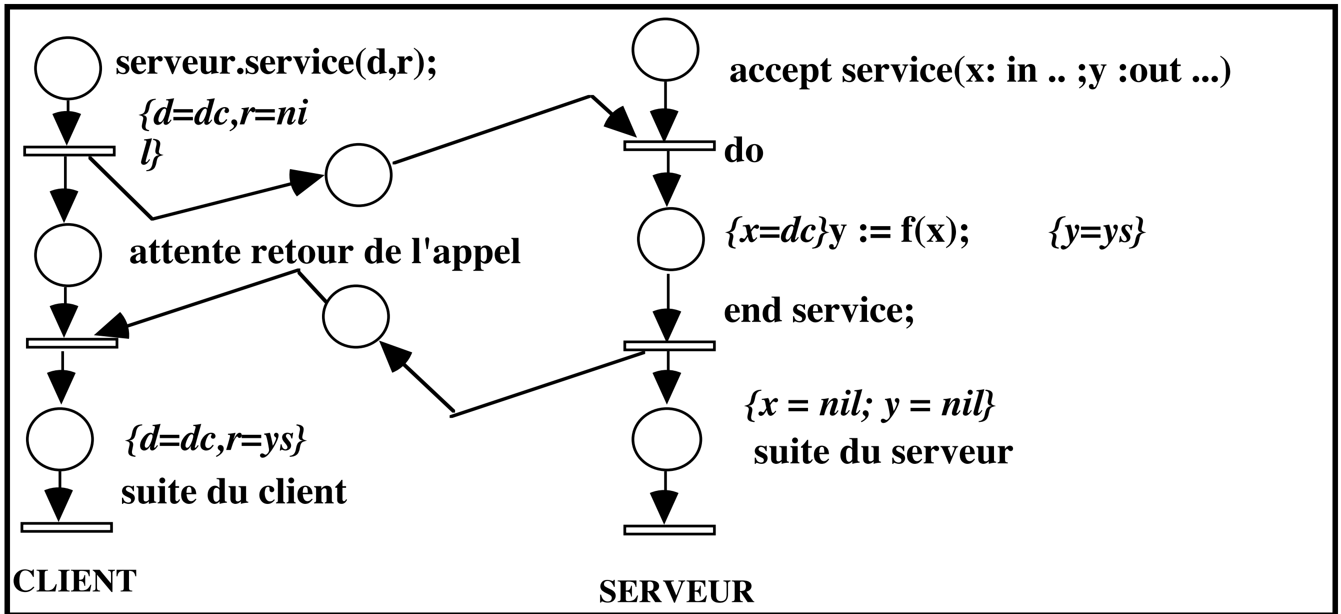
• appel avec rendez-vous conditionnel : select appel; else .... end select;

• appel avec attente limitée : select appel; or delay [until].... end select;

SEMANTIQUE DE L'INVOCATION A DISTANCE

(RENDEZ-VOUS ETENDU)

CLIENT APPELANT	**SERVEUR APPELÉ**
d : demande; r : réponse; ...	
d := calcule_la_demande; -- d= dc	
serveur.service(d,r);	**<u>accept</u> service(x: <u>in</u> demande;**
	y : <u>out</u> réponse)
	-- x = dc
	<u>do</u> y := f(x); --y = ys
	<u>end</u> service; -- x = nil; y = nil
-- d = dc, r = ys	**....**



<u>Réseau de Petri de l'invocation à distance</u>

PROCESSUS CONCURRENTS COMMUNIQUANT PAR MESSAGES ET PAR INVOCATION À DISTANCE

Exemple : le processus principal père crée deux processus fils; chaque processus lance une impression (programmation en Ada normalisé)

Code source en ADA 95 :

```
with TEXT_IO; use TEXT_IO;   -- paquetage  standard d'impression
procedure PERE is                 -- création du père, programme principal
   task type TT is                -- déclaration de l'interface du type fils
      entry START( ID : INTEGER);  -- interface de rendez-vous
   end TT;
   FILS1, FILS2 : TT;             -- création de deux fils serveurs
   task body TT is                -- déclaration du corps des fils
   begin
      accept START( ID : INTEGER) do
         PUT_LINE ("BONJOUR DU FILS "& INTEGER'IMAGE(ID));
      end START;
   end TT;
begin          -- début du père, client et activation des fils
   PUT_LINE ("BONJOUR DU PERE");
   FILS1.START(1);                -- appel et réveil du fils1 avec une requête
   FILS2.START(2);                -- appel et réveil du fils2 avec une requête
   PUT_LINE ("FIN DU PROGRAMME");
end PERE;

-- le langage Ada est normalisé ISO/IEC 8652:1995(E)
```

-- TÂCHE SERVEUR D'ACCÈS EN EXCLUSION MUTUELLE

```
package TABLEAU is         -- protège l'accès à TABLE
   procédure prendre;      -- attente si TABLEAU est déjà pris
   procédure rendre;
end TABLEAU;

package body TABLEAU is
    task serveur is
      entry pour_prendre;
      entry pour_rendre;
   end serveur;
   task body serveur is
   begin
      loop
         select
            accept pour_prendre;
            accept pour_rendre;
         or
            terminate;
         end select;
      end loop;
   end serveur ;

   procédure prendre is
   begin
      serveur.pour_prendre;
   end prendre;
   procédure rendre is
   begin
      serveur.pour_rendre;
   end rendre;
end TABLEAU;
```

-- TÂCHE SERVEUR POUR SCHÉMA PRODUCTEUR-CONSOMMATEUR

```
procedure prod-cons is
   N : constant integer := 4 ;
   task type T_STOCK is              -- spécification de la tâche serveur (ads)
      entry DEPOSER(M : in MESSAGE);
      entry RETIRER(L : out MESSAGE);
   end T_STOCK;
   task type PRODUCTEUR ; task type CONSOMMATEUR;
   TAMPON : T_STOCK;
   task body PRODUCTEUR is
      ENVOI : MESSAGE;
   begin
      ENVOI := préparation_du_message;
      TAMPON.DEPOSER(ENVOI) ;
   end PRODUCTEUR ;
   task body CONSOMMATEUR is
      RETRAIT : MESSAGE;
   begin
      TAMPON.RETIRER(RETRAIT);
      exploitation(RETRAIT);
   end CONSOMMATEUR ;
   PROD : array (1..15) of PRODUCTEUR;
   CONS : array (1..10) of CONSOMMATEUR ;

   task body T_STOCK is              -- corps de la tâche serveur (adb)
      CASIER : array(0..N-1) of MESSAGE;
      I, J : INTEGER range 0..N-1 := 0;
      NOMBRE : INTEGER range 0..N := 0:
   begin
      loop
         select
            when NOMBRE < N => accept DEPOSER(M : in MESSAGE)
               do CASIER(I) := M; end DEPOSER;
               I := (I + 1) mod N; NOMBRE := NOMBRE + 1;
         or
            when NOMBRE > 0 => accept RETIRER(L : out MESSAGE)
               do L := CASIER(J); end RETIRER;
               J := (J + 1) mod N; NOMBRE := NOMBRE - 1;
         or
            terminate;
         end select;
      end loop;
   end T_STOCK;
begin
   null;
end prod-cons ;
```

-- PRODUCTEUR CONSOMMATEUR : INTERFACE PROCEDURALE

```
package TAMPON is
   procédure déposer(x : in MESURE);-- attente si le tampon est plein
   procédure retirer(x : out MESURE);-- attente si le tampon est vide
end TAMPON;

package body TAMPON is
      task serveur is
        entry pour_déposer(x : in MESURE);
        entry pour_retirer(x : out MESURE);
   end serveur;
   task body serveur is
      NB_CASE : integer :=2;
      subtype LES_CASES is integer range 0..NB_CASE - 1;
      T : array (LES_CASES) of MESURE;
      TETE, QUEUE : LES_CASES := 0;
      NOMBRE : integer range 0 .. NB_CASE := 0;
   begin
      loop
         select
            when NOMBRE < NB_CASE =>
            accept pour_déposer(x : in MESURE)
               do T(QUEUE) := x; end pour_déposer;
               QUEUE := (QUEUE + 1) mod NB_CASE;
               NOMBRE := NOMBRE + 1;
         or
            when NOMBRE > 0 => accept pour_retirer(x : out MESURE)
               do x := T(TETE); end pour_retirer;
               TETE := (TETE + 1) mod NB_CASE;
               NOMBRE = NOMBRE - 1;
         or
            terminate;
         end select;
      end loop;
   end serveur ;
   procédure déposer(x : in MESURE) is
   begin
      serveur.pour_déposer(x);
   end déposer;
   procédure retirer(x : out MESURE) is
   begin
      serveur.pour_retirer(x);
   end retirer;
end TAMPON;
```

-- TÂCHE SERVEUR DE RENDEZ-VOUS SYMÉTRIQUE ANONYME

```
procedure E_R is
   task CANAL is
      entry EMETTRE(EMIS : in MESSAGE);
      entry RECEVOIR(RECU : out MESSAGE);
   end CANAL;

   task body CANAL is
   begin
      loop
         select
            accept EMETTRE(EMIS : in MESSAGE) do
               accept RECEVOIR(RECU : out MESSAGE) do
                     RECU := EMIS;
               end RECEVOIR;
            end EMETTRE;
         or
            terminate;
         end select;
      end loop;
   end CANAL;
   task type EMETTEUR; task type RECEPTEUR;
   task body EMETTEUR is
      ENVOI : MESSAGE;
   begin
      ENVOI := préparation_du_message;
      CANAL.EMETTRE(ENVOI) ;
   end EMETTEUR ;
   task body RECEPTEUR is
      RETRAIT : MESSAGE;
   begin
      CANAL.RECEVOIR(RETRAIT);
      exploitation(RETRAIT);
   end RECEPTEUR ;
   M : array (1..15) of EMETTEUR;
   R : array (1..10) of RECEPTEUR ;
begin
   null;
end E_R ;
-- chaque EMETTEUR communique avec n'importe quel RECEPTEUR
-- la tâche CANAL le met en communication anonyme directe
```

-- ATTENTE SÉLECTIVE COTÉ SERVEUR APPELÉ

-- ATTENTE LIMITÉE DANS LE TEMPS

```
select
   accept SIGNAL_DE_VEILLE;
or
   delay 30.0;       -- variante : delay until ECHEANCE;
   ARRETER_LE_TRAIN;  -- arrêt d'urgence
end select;
```

-- RENDEZ-VOUS CONDITIONNEL (OU IMMÉDIAT)

```
select
   accept RENDRE
      do .... end RENDRE;
else
   PUT( "pas de retour de ressource en ce moment");
end select;
```

-- APPEL D'ENTRÉE CONDITIONNEL COTÉ APPELANTE

-- DEMANDE DE RENDEZ-VOUS CONDITIONNEL (OU IMMÉDIAT)

```
select
   R1.PRENDRE;
   PUT("R1 acquise");
else
   PUT("R1 non acquise");
end select;
```

-- APPEL D'ENTRÉE À ATTENTE LIMITÉE DANS LE TEMPS

```
select
   R2.PRENDRE;
   PUT("R2 acquise");
or
   delay 20.0;          -- variante : delay until ECHÉANCE
   PUT("R2 non acquise");
end select;
```

```
-- ALLOCATION D'UNE CLASSE DE RESSOURCES
with Text_Io; use Text_Io;
with Ada.Numerics.Float_Random; -- fournit le tirage aleatoire
use Ada.Numerics.Float_Random;
with resource_management ; use resource_management ; -- premier
-- with resource_control ; use resource_control ; -- second exemple
with Common; use Common; -- fournit les valeurs communes

procedure resource_allocation is -- PROGRAMME PRINCIPAL

   task type user_type(x : r_number := r_number'first);
            -- la valeur par défaut de x est r_number'first recue a la creation

   task body user_type is
      R : resource ;
   begin
      put_line("        un client demande " & r_number 'image(x));
      controller.request(R, x);
      delay 1.0;
      put_line("        un client restitue " & r_number 'image(x));
      controller.release(R, x);
   end user_type ;

   G : Generator; -- loi uniforme de 0.0 à 1.0

   user : array(1..100) of user_type
                  (x => r_number(Max(1.0, Random(G)*Max)));
         -- la demande x est un entier compris entre 1 et Max
         -- le premier Max est une fonction, le deuxième est une constante
begin -- on a créé cent tâches et elles sont activées
   null;
end resource_allocation ;
-- ==================================================
```

```
-- ALLOCATION D'UNE CLASSE DE RESSOURCES
package Common is
   Max : constant integer := 100; -- number of resources
   type r_number is range -Max .. Max;
   type resource is new Integer;
end Common;


-- SERVICE AU MIEUX ---- premier exemple
with Text_Io; use Text_Io;
with Common; use Common; -- fournit les valeurs communes
package resource_management is

   task controller is
      entry request(R : out resource; number : r_number);
      entry release(R : in resource; number : r_number);
   private
      entry assign(R : out resource; number : r_number);
   end controller ;

end resource_management ; -- fin du paquetage de spécification
```

-- SERVICE AU MIEUX ---- premier exemple

```
package body resource_management is
   task body controller is
      free : r_number := r_number'last ;
      to_try : natural := 0 ; new_resources_released : boolean := false ;
   begin
      loop
         select
            when free > 0 =>
               accept request (R : out resource; number : r_number) do
                  if number <= free then
                     free := free - number ; -- allocate_resources(R)
                     put_line("don de " & r_number 'image(number));
                  else
                     put_line("attente de " & r_number 'image(number));
                     requeue assign ;
                  end if;
               end request ;
         or
            when new_resources_released =>
               accept assign(R : out resource; number : r_number) do
                  to_try := to_try - 1 ;
                  if to_try = 0 then
                     new_resources_released := false;
                  end if;
                  if number <= free then
                     free := free - number ; -- allocate_resources(R)
                     put_line("don de " & r_number 'image(number));
                     -- si on ne doit pas garder un reexamen FIFO
                     if free = 0 then -- si reexamen en tourniquet
                        new_resources_released := false;
                     end if;
                        -- on arrete le reexamen des demandes
                        -- au prochain release le suivant de la file est premier
                  else
                     put_line("attente de " & r_number 'image(number));
                     requeue assign ;
                  end if;
               end assign ;
```

```
        or
           accept release(R : in resource; number : r_number) do
              free := free + number ;  -- free resources
              put_line("retour de " & r_number 'image(number));
              to_try := assign'count;
              new_resources_released := (to_try > 0) and (free > 0);
              -- un client peut rendre 0 ressources - inutile de reveiller
           end release;
        or
           terminate;
        end select;
     end loop;
  end controller ;

end resource_management ; -- fin du paquetage de description du corps
```

-- service au mieux; il peut y avoir famine des gros demandeurs
-- deux cas : le réexamen des requêtes en attente est
--soit FIFO et on recommence toujours par la plus ancienne
-- soit en tourniquet et on recommence là où on s'est arrêté

-- ALLOCATION D'UNE CLASSE DE RESSOURCE

```
with Text_Io; use Text_Io;
with Common; use Common; -- fournit les valeurs communes
package resource_control is

                                   -- SERVICE FIFO

  task controller is
    entry request(R : out resource; number : r_number);
    entry release(R : in resource; number : r_number);
  private
    entry assign(R : out resource; number : r_number);
  end controller ;

end resource_control ;
```

```
-- SERVICE FIFO : service selon l'ordre d'arrivee sur l'entree request
-- une seule tache attend sur l'entree assign et bloque les request
package body resource_control is
   task body controller is
      free : r_number := r_number’last ;
   begin
      loop
         select
            when assign’count = 0 =>
               accept request (R : out resource; number : r_number) do
                  free := free - number ;-- note la requete, alloue si possible
                  if free < 0 then
                     put_line("attente de " & r_number 'image(number));
                     requeue assign ;
                  else
                  -- allocate_requested_resources(R) ;
                     put_line("don de " & r_number 'image(number));
                  end if;
               end request ;
         or
            when free >= 0 =>
               accept assign(R : out resource; number : r_number) do
                  -- allocate_requested_resources(R) ;
                     put_line("don de " & r_number 'image(number));
               end assign;
         or
            accept release(R : in resource; number : r_number) do
               free := free + number ;  -- records released resources
               -- free_released_resources(R);
               put_line("retour de " & r_number 'image(number));
            end release;
         or
            terminate;
         end select;
      end loop;
   end controller ;
end resource_control;
```

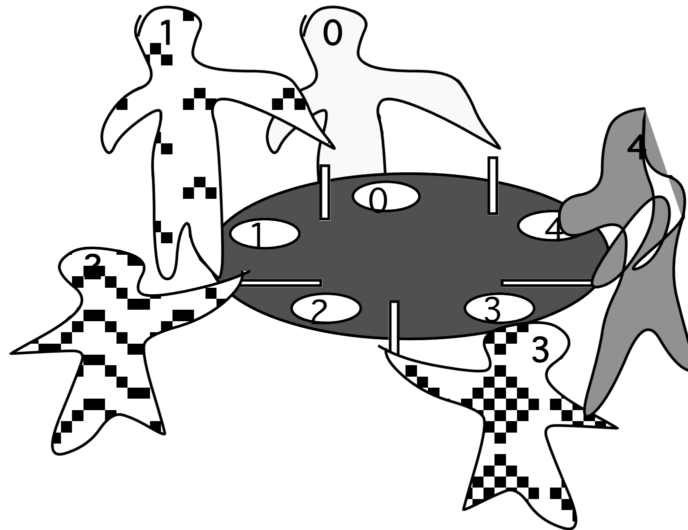
REPAS DES PHILOSOPHES (REPAS ASSIS)
protocole externe

```
procedure philosophes_assis is   -- solution avec N-1 chaises
   N : constant integer := 5;
   task type ressource is entry prendre; entry rendre; end ressource;
   baguette : array (1..N) of ressource;
   task chaise is entry prendre; entry rendre; end chaise;
   task numéro is entry unique(id : out integer); end numéro; -- 1 à N
   task type philo;
   philosophe: array (1..N) of philo;
   task body philo is gauche, droite : integer;
   begin
      numéro.unique(gauche);
      if gauche = N then droite := 1; else droite := gauche + 1; end if;
      for I in (1.. 57) loop
         put(gauche); put(" pense"); new_line;
         delay(duration(gauche * droite)); -- il pense un certain temps
         put(gauche); put(" a faim"); new_line;
         chaise.prendre; -- pour s'asseoir
         baguette(gauche).prendre;
         baguette(droite).prendre;
         put(gauche); put("mange"); new_line;
         delay(duration(gauche * gauche)); -- il mange un certain temps
         baguette(gauche).rendre;
         baguette(droite).rendre;
         chaise.rendre;
      end loop;
   end philo;

   task body ressource is separate;
   task body chaise is separate;
   task body numéro is separate;

begin
   null;
end philosophes_assis;
```

-- REPAS DES PHILOSOPHES (REPAS ASSIS)

```
separate (philosophes_assis)
task body ressource is
begin
   loop
       select
           accept prendre; accept rendre;
       or
           terminate;
       end select;
   end loop;
end ressource;
separate (philosophes_assis)
task body chaise is
   libre : integer := N - 1;
begin
   loop
       select
           when libre > 0 => accept prendre; libre := libre - 1;
       or
           accept rendre; libre := libre + 1;
           -- on pourrait lever une exception quand on a déjà N-1 chaises
       or
           terminate;
       end select;
   end loop;
end chaise;
separate (philosophes_assis)
task body numéro is I : integer := 1; --début de numérotation à 1
begin
   loop
       select
           accept unique(id : out integer) do id := I; end unique;
           I := I + 1;
       or
           terminate;
       end select;
   end loop;
end numéro;
```

REPAS DES PHILOSOPHES
protocole interne



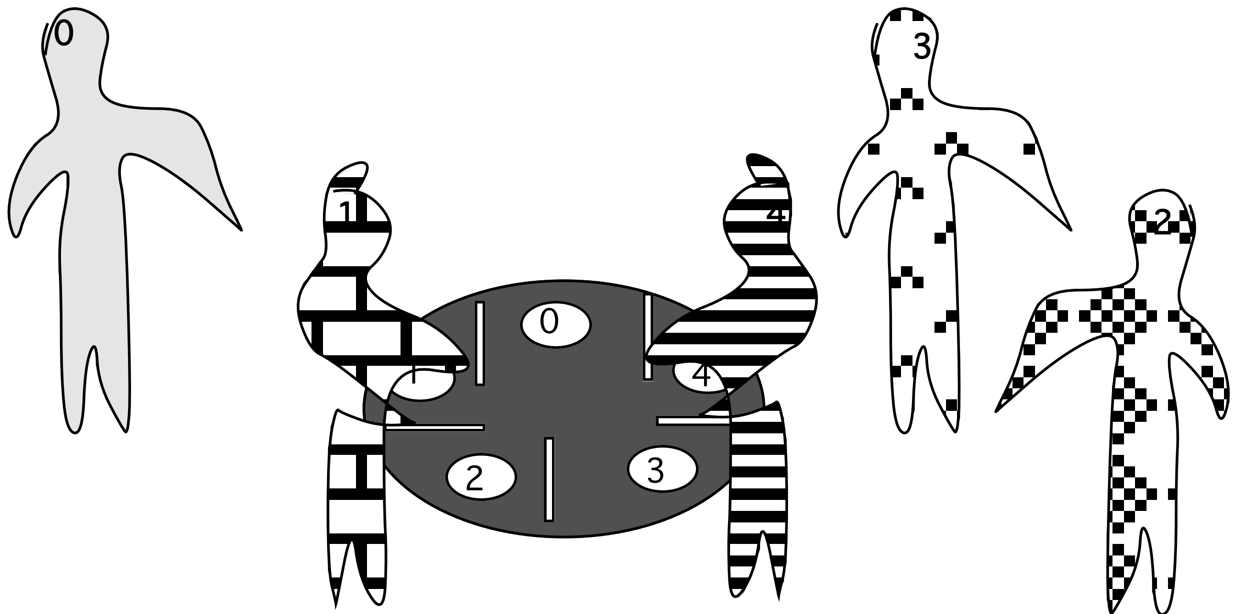
allocation une baguette après l'autre
protocole interne

interblocage !!

```
package Chopsticks_Server is
   procedure request(me : in philo_id) ;
   procedure release(me : in philo_id) ;
end Chopsticks_Server ; --end of the package specification (ads)
```

```
package body Chopsticks_Server is --REPAS DES PHILOSOPHES
   task server is
    entry get_pair_0; entry get_pair_1; entry get_pair_2;
    entry get_pair_3; entry get_pair_4;
    entry release_pair(x : in philo_id);
   private entry finish_pair_0; entry finish_pair_1;
    entry finish_pair_2; entry finish_pair_3; entry finish_pair_4;
   end server;
   procedure request(me : in philo_id) is
   begin
       case me is
           when 0 =>server.get_pair_0; when 1 =>server.get_pair_1;
           when 2 =>server.get_pair_2; when 3 =>server.get_pair_3;
           when 4 =>server.get_pair_4;
       end case;
   end request;
   procedure release(me : in philo_id) is
   begin server.release_pair(me); end release;
   task body server is
    available : boolean_array := (others => true); --chopsticks
   begin
      loop
         select
            when available(0) accept get_pair_0 do
                  available(0) := false; requeue finish_pair_1 ;
            end get_pair_0;
         or
            when available(0) accept finish_pair_0 do
                  available(0) := false;
         end finish_pair_0 ;
         or .......
            when available(4) accept get_pair_4 do
                  available(4) := false; requeue finish_pair_0 ;
            end get_pair_4;
         or
            when available(4) accept finish_pair_4 do
                  available(4) := false;
         end finish_pair_4 ;
         or
            accept release_pair(x : in philo_id) do
               available(x) := true; available(x + 1) := true;
            end release_pair;
         or
            terminate;
         end select;
      end loop;
   end server ;
end Chopsticks_Server ; --end of the package body (adb)
```

REPAS DES PHILOSOPHES
protocole interne



allocation globale

protocole interne

available (x) and available(x + 1)
famine

available (x) and available(x + 1) and not requestor(x - 1)
ni famine ni interblocage
(Courtois, Georges, On starvation prevention, RAIRO vol 11, 1977)

```
package Task_Server_Chops is
   procedure request(me : in philo_id) ;
   procedure release(me : in philo_id) ;
end Task_Server_Chops ; --end of the package specification (ads)
```

```
package body Task_Server_Chops is --REPAS DES PHILOSOPHES
   task server is
    entry get_pair(x :in philo_id);
    entry release_pair(x : in philo_id);
   private entry please_get_pair(x : in philo_id);
   end server;
   procedure request(me : in philo_id) is
   begin server.get_pair(me); end request;
   procedure release(me : in philo_id) is
   begin server.release_pair(me ); end release;
   task body server is
    flush_count : integer := 0;
    available : boolean_array := (others => true); --chopsticks
    requestor : boolean_array := (others => false); -- philosophers
   begin
      loop
         select
            accept get_pair(x : in philo_id) do
      if available(x) and not requestor(x - 1) and available(x + 1) then
                    available(x) := false; available(x + 1) := false;
      else
                    requestor(x) := true;
                    requeue please_get_pair ;
      end if;
            end get_pair;
         or
            when flush_count > 0 =>
            accept please_get_pair(x : in philo_id) do
               flush_count := flush_count - 1;
         if available(x) and not requestor(x - 1) and available(x + 1) then
                    available(x) := false; available(x + 1) := false;
                    requestor(x ) := false;
         else
                    requeue please_get_pair ;
         end if;
            end please_get_pair;
         or
            accept release_pair(x : in philo_id) do
               available(x) := true; available(x + 1) := true;
               flush_count := please_get_pair'count;
            end release_pair;
         or
            terminate;
         end select;
      end loop;
   end server ;
end Task_Server_Chops ; --end of the package body
-- exercice, prouver l'absence d'interblocage
```

-- REPAS DES PHILOSOPHES
-- Programme principal des processus philosophes

```
procedure Application is

   next_id : philo_id := philo_id'first;
   function unique_id return philo_id is
   begin
      next_id := next_id + 1 ; -- addition modulo N
      return next_id ;
   end unique_id;

   task type philo(x : philo_id  := unique_id) ;
   philosophe : array(philo_id) of philo ;

   -- avec une tâche serveur pour le contrôle de l'utiisation des baguettes
   use Task_Server_Chops ;
      -- ou bien use Chopsticks_Server;
   -- avec une objet protégé pour le contrôle de l'utiisation des baguettes
      -- use Protected_Object_Chops;
            -- ou encore use chopsticks_object;

   task body philo is
   begin
      loop
         thinking;
         request(x) ;
         eating ;
         release(x) ;
      end loop ;
   end philo ;

end Application ;
```