ACCOV 2001-2002. Système avec des processus concurrents

procedure Main is

type TabDoc is array(1..6) of Document;

```
-- contrôle l'attribution d'un nom unique
package ProcId is
   procedure PrendreUnique(
                              I : out Integer);
end ProcId;
```

```
-- contrôle l'ordre de mise en route
package Annonce is
   procedure Attendre;
   procedure Signaler;
   procedure LancerTous;
   procedure AttendreLaFindeTous;
end Annonce;
```

```
-- contrôle l'allocation de ressources
package Ressource is
   -- X ressources au processus de nom I
   procedure Allouer(
                 X : in Integer; I : in Integer);
   -- Y ressources rendues par I
   procedure Restituer(
                 Y : in Integer; I : in Integer);
end Ressource;
```

```
-- contrôle l'utilisation du canal partagé - Q3
package Canal is
   procedure Ouvrir(I : in Integer);
   procedure Fermer(I : in Integer);
end Canal;
```

```
-- déclaration du type Acteur

task type TypeActeur is
   -- nom unique du processus Acteur
   Id : Integer; -- NomUnique de 1 à 4
   --document établi par l'exécution de C
   MonCompteRendu : Document;
   K, Y : Integer ; -- nombres de ressources
   begin

   -- attribuer une valeur à Id
   ProcId.PrendreUnique(Id);

   -- attendre le signal pour imprimer
   Annonce.Attendre;
   -- afficher son démarrage et son nom
   Put_Line(
      "Demarrage de " & Integer'Image(Id));
   -- annoncer qu'il a fini d'imprimer
   Annonce.Signaler;
   Y := 0; -- total de ressources reçues

   for X in 1..3 loop
      K := tirage(1,5); -- aléatoire 1 ≤ K ≤ 5
      -- prendre K ressources de plus
      Ressource.Allouer(K, Id);
      Y := Y + K;
      C; --execution du code specifique
      -- avec les Y ressources reçues
      Canal.Ouvrir(Id);
      Diffuser(MonCompteRendu); -- réseau
      Canal.Fermer(Id);
   end loop;

   -- rendre toutes les ressources reçues
   Ressource.Restituer(Y, Id);

end TypeActeur;
```

```
-- déclaration, création de 4 acteurs
A : array(1..4) of TypeActeur;
```

```
-- code du processus Main

   Id : Integer := 0; -- nom donné à Main
begin -- à ce point, les 5 tâches démarrent
   -- afficher son démarrage et son nom
   Put_Line("Demarrage de Main" );
   --autoriser les autres annonces
   Annonce.LancerTous;
   -- attendre la fin des annonces
   Annonce.AttendreLaFinDeTous;
   Put_line("Fin du demarrage");

end Main;
```

Système avec des processus concurrents

On étudie quelques aspects du système suivant qui comprend 5 processus concurrents : le processus Main et 4 processus Acteurs, et des paquetages de contrôle de concurrence.

Les processus sont concurrents et les 5 processus démarrent en même temps. Chaque processus qui démarre, commence par demander l'impression d'une annonce. On suppose que chaque impression est une action atomique.

Chaque Acteur va exécuter un calcul compliqué, C, pour lequel il lui faut d'abord obtenir des ressources, en nombre aléatoire (chaque demande nouvelle est comprise entre 1 et 5). A chaque cycle de calcul, l'Acteur produit un document qu'il diffuse sur le réseau local. Pour cela il faut qu'il ait accès au canal de diffusion.

Le schéma du système a été donné page 1.

1. ORDRE POUR IMPRIMER LES ANNONCES DE DÉMARRAGE

Chaque processus qui démarre commence par demander l'impression d'une annonce. On suppose que chaque impression Put_Line est une action atomique (l'exclusion mutuelle est gérée par le sous-système d'entrée-sortie). Les processus sont concurrents et les 5 processus démarrent en même temps.

Première politique. (Main d'abord, puis les 4 acteurs dans le désordre)

Le processus Main va orchestrer les impressions d'annonces de la façon suivante.

Chaque processus, sauf Main, commence par attendre le signal qui l'autorise à demander son annonce. Puis, quand il a reçu ce signal (qui est envoyé par Main), il imprime son annonce, et il envoie à son tour un signal vers Main.

De son coté, Main va les réveiller tous ensemble en envoyant à tous le signal qu'ils attendent. Puis Main attendra que tous les 4 autres processus lui signalent leur fin d'impression.

Le paquetage Annonce fournit 4 procédures pour cette coordination :
- La procédure Attendre permet de bloquer le processus qui appelle cette procédure.
- La procédure LancerTous permet de libérer les 4 processus bloqués par Attendre ou en passe de l'être.
- La procédure Signaler permet d'envoyer un signal vers Main.
- La procédure AttendreLaFinDeTous permet à Main d'attendre les 4 signaux des autres processus.

Solution avec des sémaphores

```
package body Annonce is
   -- déclaration de sémaphores et de variables d'état
MainAAutres, AutresAMain : Semaphore;

   procedure Attendre is
   begin
      -- bloquer le processus appelant tant que Main n'a pas exécuté LancerTous
      P(MainAAutres);
   end Attendre;

   procedure Signaler is
   begin
      -- informer le processus Main que l'appelant a fini
      V(AutresAMain );
   end Signaler;

   procedure LancerTous is
```

```
   begin
      -- envoyer aux 4 processus le signal qui leur permet d'imprimer leur annonce
      for I in 1..4 loop V(MainAAutres); end loop;
   end LancerTous ;
   procedure AttendreLaFinDeTous is
   begin
      -- bloquer le processus Main tant que tous les 4 autres processus n'ont pas fait signe de vie
      for I in 1..4 loop P(AutresAMain ); end loop;
   end AttendreLaFinDeTous ;

begin
   -- initialiser chaque sémaphore
   E0(AutresAMain, 0); E0(MainAAutres, 0);
end Annonce;
```

Deuxième politique (Main d'abord, puis les 4 acteurs dans un ordre imposé) (non programmée page 1)

Les 5 processus ont des noms de 0 à 4. En effet, on a donné les noms 0 à Main et les 4 Acteurs prennent dynamiquement un nom unique, de 1 à 4.

On va utiliser cet ordre de numérotation pour imposer un ordre d'impression pour les annonces de démarrage. Pour cela, chaque processus J, sauf Main, commence par attendre son tour. Puis, quand il a reçu le signal de réveil envoyé par son prédécesseur, il imprime son annonce, et il envoie à son tour un signal de réveil à son successeur (J + 1) modulo 6. L'acteur qui a le nom unique 5 envoie un signal vers Main.

Dans le paquetage Depart, on utilise un tableau SemBloc de 5 sémaphores, qui servent chacun à contrôler un processus. Ainsi dans Attendre(I), SemBloc(I) sert à bloquer le processus de nom unique I. Dans Signaler(J), on utilise Sembloc((J + 1) mod 5) pour réveiller le processus (J + 1) mod 5.

```
package body Depart is
   SemBloc : array(0..4) of semaphore;
   procedure Attendre(I : in Integer) is
   -- bloquer le processus de nom I
   begin   P(SemBloc(I));     end Attendre;
   procedure Signaler(J : in Integer) is
   -- réveiller le processus successeur de nom (J + 1) mod 5
   begin   V(SemBloc((J +1) mod 5)) ;   end Signaler;
begin
   -- initialiser chaque sémaphore du tableau SemBloc
   for K in 0..4 loop
       E0(SemBloc(K), 0);   -- initialiser SemBloc(K);
   end loop;
end Depart;
```

Troisième politique (Les 4 acteurs dans le désordre puis Main) (non programmée page 1)

On veut autoriser les 4 acteurs à démarrer en parallèle sans leur imposer d'ordre. On va maintenant les réveiller tous ensemble et on ne réveillera Main que lorsqu'ils auront tous signalé leur fin d'impression.

Autres politiques : anneau virtuel, politique selon votre invention

2. ALLOCATION DES RESSOURCES SANS INTERBLOCAGE

Première politique (précaution préalable contre l'interblocage)

On suppose qu'il y a au moins 45 ressources. Montrer qu'il ne peut jamais avoir interblocage et qu'on est dans le cas de la prévention statique par précaution préalable. On alloue alors les ressources sans contrôle.

Deuxiéme politique (prévention de l'interblocage)

On suppose qu'il y a moins de 45 ressources (ou on prépare une évolution future de la demande) et on met en place l'algorithme du banquier, en supposant que l'annonce est de 15.

Algorithme du banquier selon le cours, avec une classe de ressource et des annonces toutes égales. On doit toujours garder assez de ressources pour servir le processus leader, celui qui est le plus proche de son annonce. Il faut noter les allocations déjà faites aux 4 acteurs, vérifier que le leader peut être servi avec le reste du résidu, une fois l'allocation demandée faite.

```
Soit     R : Integer; -- résidu de ressources
         Total : array(1..4) of Integer; -- allocation faite aux 4 acteurs
      -- le processus I demande X ressources en plus et son annonce est A
function EtatFiable(X : in Integer; I : in Integer) return Boolean is
   OK : Boolean;  -- on peut servir le leader
   PourVoir : Integer; -- pour voir si le leader pourra être servi avec le reste du résidu
begin
   If R < X then
      return False ;
   else
      R := R - X; -- -- on attribue les X ressources pour voir
      Total(I) := Total(I) + X; -- on attribue les X ressources à I, pour voir
      for J in 1..4 loop
         PourVoir := Total(J) + R; -- on ajoute le résidu pour voir si
         OK := (PourVoir >= A); -- un acteur au moins pourra obtenir son annonce
         exit when OK; -- on en a trouvé 1, c'est bon
      end loop;
      R := R + X;  -- on a vu, on revient à l'état initial pour R et Total
      Total(I) := Total(I) - X;
      return OK;
   end if;
end EtatFiable;
```

3. ACCÈS AU CANAL PARTAGÉ

Pour diffuser chaque document sur le réseau, chaque acteur doit avoir accès au canal. L'accès au canal partagé doit se faire en exclusion mutuelle entre les acteurs. Si on utilise classiquement un sémaphore d'exclusion mutuelle, on n'est pas maître de l'ordre des réveils en cas d'attente, car l'ordre de réveil dépend de l'ordre de la file d'attente du sémaphore, et cet ordre peut varier d'un système à un autre.

La fonction ChoixCandidat élit un candidat selon la politique de réveil qui est programmée.

Choix équitable. (pas de famine) Corrigé avec des sémaphores

Pour pouvoir gérer explicitement l'ordre des réveils, on va utiliser le schéma des sémaphores privés et des variables d'état :

- une variable booléenne Libre pour indiquer que le canal est disponible,
- une variable booléenne Attente pour indiquer qu'il y a au moins un processus en attente
- un tableau de booléens Attend() pour indiquer quels sont les processus qui attendent.

```
package body Canal is

   Libre : Boolean := True ; -- le canal est disponible quand Libre = True
   Attend : array(1..4) of Boolean:= False;  -- le processus I est bloqué quand Attend(I) = True
   Attente : Boolean := False; -- quand Attente = True, il y a au moins un processus en attente
   Suivant : Integer := 4;  -- nom d'un processus, index de parcours du tableau Attend
   -- déclaration des sémaphores utilisés
Mutex: Semaphore; Sempriv : array(1..4) of Semaphore;
   procedure Ouvrir(I : in Integer) is
      -- le processus de nom I demande l'accès exclusif au canal
      OK : Boolean; -- accès possible ou non
   begin
      P(Mutex);
      OK := Libre;
      if OK then
         Libre := False;  -- I prend le droit d'utiliser le canal
         -- décider qu'il faut autoriser I à continuer son exécution
         V(Sempriv(I));
      else
         Attente := True; Attend(I) := True;
         -- décider qu'il faut bloquer I en attente de ressource
      end if;
      V(Mutex);
      P(Sempriv(I));  -- la décision est appliquée à ce niveau, hors section critique
   end Ouvrir;

   procedure Fermer(I : in Integer) is
      J : Integer;  -- nom du processus à réveiller
      function ChoixCandidat return Integer is -- choisit un processus en attente
      begin
         loop
            if Suivant = 4 then Suivant := 1 else Suivant := Suivant + 1; end if;
            exit when Attend(Suivant); -- sortie de la boucle  quand Attend(Suivant) = True
         end loop;
         return Suivant;
      end ChoixCandidat ;
   begin
      P(Mutex);
      Libre := True; -- libération du canal par le processus I
      if Attente then
         J := ChoixCandidat; Attend(J) := False; Libre := False;
         -- réveiller le processus J
         V(Sempriv(J));
         -- voir s'il y a au moins un autre processus en attente et le noter dans Attente
         for K in 1..4 loop Attente := Attend(K); exit when Attente; end loop;
      end if;
      V(Mutex);
   end Fermer;

begin            -- initialisation des sémaphores utilisés
   E0(Mutex, 1); for K in 1..4 loop E0(Sempriv(K), 0); end loop;
end Canal ;
```

Choix selon priorités. (famine possible) Corrigé avec des sémaphores

Il suffit, dans ChoixCandidat de commencer la recherche du suivant toujours à partir de 1. Alors Suivant n'est plus une variable globale ; ChoixCandidat devient :

```
function ChoixCandidat return Integer is -- choisit un processus en attente
   Suivant : Integer;
begin
   for Suivant in 1..4 loop
      exit when Attend(Suivant);
   end loop;
   return Suivant;
end ChoixCandidat ;
```

Le reste demeure inchangé.

Autres algorithmes possibles.

1. Autres algorithmes d'exclusion mutuelle (algorithmes répartis): avec anneau virtuel, avec arborescence logique (Naïmi Trehel), avec estampilles (Lamport, Ricart-Agravala)
2. Algorithmes de diffusion ordonnée sans exclusion mutuelle mais en numérotant, à l'émission, les messages diffusés et en délivrant, à la réception, les messages selon l'ordre des numéros. La numérotation peut être obtenue avec un séquenceur centralisé ou réparti, avec une exclusion mutuelle, avec un anneau virtuel,....