CORRIGÉS DE L'EXAMEN DU 16 JUIN 2001

portant sur l'enseignement des SYSTÈMES INFORMATIQUES

UTILISATION DE LA RESSOURCE UNITÉ CENTRALE

Solution 1.1. : Trace UC1

1) date	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
prêt attente UC ancienneté		T2	T2	T2, T3	T3	T3, T4	T3,, T4	T3, T4, T5	T4, T5	T4, T5	T5	T5	T5	T5		
élu	T1	T1	T1	T1	T2	T2	T2	T2	T3	T3	T4	T4	T4	T4	T5	T5

Solution 1.2.: Trace UC2

2) date	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
prêt attente UC tourniquet		T2	T1	T1, T3	T3 T2	T3, T2, T4	T2 T4	T2, T4, T5	T4 G	T4, T5	T5	T5	T4	T4		
élu	T1	T1	T2	T2	T1	T1	T3	T3	T2	T2	T4	T4	T5	T5	T4	T4

Le temps de réponse moyen est plus mauvais avec le tourniquet car on a des travaux qui sont longs par rapport au quantum. On constate que le tourniquet favorise bien les travaux courts T3 et T5.

Solution 1.3. : Trace UC3

3) date	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
prêt attente UC tourniquet		T2	T1	T1, T3	T3 T2	T3, T2, T4	T2 T4	T2, T4, T5	T4 G	T5 T3	T5 T3	T3 T4	T3 T4	T4		
élu	T1	T1	T2	T2	T1	T1	T3	T2	T2	T4	T4	T5	T5	T3	T4	T4
S. critique				T2				T2	T2					T3		

Solution 1.4.: TraceUC4

4) date	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
prêt attente UC		T1	T1	T2 T1	T1	T2, T1	T2 T1	T4, T2, T1	T4 T2 T1	T2 T1	T2 T1	T1	T1			
priorités	T1	T2	T2	T3	T2	T4	T4	T5	T5	T4	T4	T2	T3	T1	T1	T1
S. critique			T2		T2							T2	T3			

Avec section critique, pour T3 il y a inversion de l'ancienneté et inversion de priorité. Pour éviter ce dernier cas, il faut mettre l'héritage de priorité (ou la priorité plafond). On aura alors la séquence T1 T2 T2 T3 T2 T2 T3...T3 se terminera en 7, avec un temps de réponse de 4. (exécution T3 + fin de la section critique de T2)

Synthèse des Temps de réponse

				FIFO	FIFO	tourn	iquet	tourn	+ SC	prio	+ SC
	date réveil	durée d'exec	prio rité	date de fin	temps réponse	date de fin	temps réponse	date de fin	temps réponse	date de fin	temps réponse
T1	0	4	5	4	4	6	6	6	6	16	16
T2	1	4	4	8	7	10	9	9	8	12	11
T3	3	2	1	10	7	8	5	14	11	13	10
T4	5	4	3	14	9	16	11	16	11	11	6
T5	7	2	2	16	9	14	7	13	6	9	2
mo	yen	ne			7,2		7,6		8,4		9,0

2. ORDRE POUR IMPRIMER LES ANNONCES DE DÉMARRAGE

Ordre 1. Question 2.1.

```
package body Depart is
   SemBloc : array(0..5) 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 6
   begin    V(SemBloc((J +1) mod 6)) ;    end Signaler;
begin
   -- initialiser chaque sémaphore du tableau SemBloc
   for K in 0..5 loop
       E0(SemBloc(K), 0);   -- initialiser SemBloc(K);
   end loop;
end Depart;
```

Ordre 2. Question 2.2.

```
package body Depart is
   SemBloc : array(0..5) of semaphore;
   Mutex : Semaphore;
   Comptage : Integer := 0;
   procedure Attendre(I : in Integer) is
   begin
      P(SemBloc(I));   -- bloquer le processus de nom I
   end Attendre;
   procedure Signaler(J : in Integer) is
   begin
      case J is
         when 0 => V(SemBloc(1); -- réveiller le processus de nom 1;
         when 1 => -- réveiller les processus de noms 2, 3, 4 et 5;
                     V(SemBloc(2); V(SemBloc(3); V(SemBloc(4); V(SemBloc(5);
         when 2..5 =>  -- quand J vaut une des valeurs de 2 à 5, on fait ce qui suit
                        P(Mutex);
                        Comptage := Comptage + 1;
                        if Comptage = 4 then V(SemBloc(0); end if;
                        V(Mutex);
      end case
   end Signaler;
begin
   E0(Mutex, 1);
   -- initialiser chaque sémaphore du tableau SemBloc
   for K in 0..5 loop
       E0(SemBloc(K), 0);   -- initialiser SemBloc(K);
   end loop;
end Depart;
```

3. GESTION DE L'ALLOCATION DE RESSOURCES

Solution 3.1. Sémaphores privés

```
package body Ressource is
   R : Integer := N; -- repére les ressources disponibles. Le résidu initial vaut N
   Total : array(2..5) of Integer := 0; -- allocation initiale nulle
   Attend : array(2..5) of Boolean:= False; Attente : Boolean := False; -- gestion de l'attente
   Combien : array(2..5) of Integer:= 0; -- demande nouvelle nulle initialement
   -- déclaration des sémaphores utilisés
   Mutex : Semaphore; Sempriv : array(2..5) of semaphore;
   procedure Allouer(X : in Integer; I : in Integer) is
      OK : Boolean; -- allocation possible ou non
   begin
      P(Mutex);
      OK := (R>= X);
      if OK then
         R := R - X; -- allocation des ressources à I
```

```
            Total(I) := Total(I) + X;
            -- autoriser I à continuer son exécution
            V(Sempriv(I));
         else
            Attente := True; Attend(I) := True; Combien(I) := X;
            -- bloquer I en attente de ressource
         end if;
         V(Mutex);
         P(Sempriv(I));
      end Allouer;
      procedure Restituer(Y : in Integer; I : in Integer) is
         J : Integer;  -- nom d'un processus bloqué
         function ChoixCandidat return Integer; -- choisit un processus en attente
      begin
         P(Mutex);
         R := R + Y; Total(I) := Total(I) - Y;
         While Attente loop
            J := ChoixCandidat;
            exit when R < Combien(J); -- pas assez de ressource, on arrête les réveils
            Attend(J) := False; R := R - Combien(J); Combien(J) := 0; Attente := False;
            -- réveiller le processus J
            V(Sempriv(J));
            -- voir s'il y a au moins un autre processus en attente
            for K in 2..5 loop Attente := Attend(K); exit when Attente; end loop;
         end loop;
         V(Mutex);
      end Restituer;
   begin
   -- initialisation des sémaphores utilisés
      E0(Mutex, 1); for I in 2..5 loop E0(Sempriv(I), 0); end loop;
   end Ressource;
```

Question 3.2. une seule requête en attente

```
   package body Ressource is
      R : Integer := N; -- repére les ressources disponibles. Le résidu initial vaut N
      Total : array(2..5) of Integer := 0; -- allocation initiale nulle
      Attente : Boolean := False; -- gestion de l'attente
      SemCandidat : Semaphore;
      -- déclaration des autres sémaphores utilisés
      MutexService, Mutex : Semaphore;
      procedure Allouer(X : in Integer; I : in Integer) is
      begin
         -- entrée d'exclusion mutuelle entre les processus qui appellent Allouer
         P(MutexService); P(Mutex); -- dans cet ordre pour les sémaphores
         R := R - X; --allocation des ressources à I, allocation immédiate ou prévue
         Total(I) := Total(I) + X;
         if R >= 0 then
            Attente := False;
            -- autoriser I à continuer son exécution
            V(SemCandidat);
         else
            Attente := True;
            -- bloquer I en attente de ressource en utilisant SemCandidat
         end if;
         V(Mutex); P(SemCandidat); V(MutexService);
         -- sortie d'exclusion mutuelle entre les processus qui appellent Allouer
      end Allouer;
      procedure Restituer(Y : in Integer; I : in Integer) is
      begin
```

```
      P(Mutex);
      R := R + Y; Total(I) := Total(I) - Y;
      If Attente and R >=0 then
         Attente := False;
         -- réveiller le processus bloqué par SemCandidat
         V(SemCandidat);
      end if;
      V(Mutex);
   end Restituer;
begin
-- initialisation des sémaphores utilisés
   E0(Mutex, 1); E0(MutexService 1); E0(SemCandidat, 0);
end Ressource;
```

Solutions Interblocage

Question 3.3. - Exemple d'interblocage. Trois acteurs ont 4 ressources, le quatrième en a deux. Aucune nouvelle demande ne peut être satisfaite et quand les 4 acteurs auront fait leur nouvelle demande, le système sera en interblocage.

***Question* 3.4. - a)** Il faut 6*4 +1 ressources = 25 ressources pour qu'un acteur puisse être servi dans le pire cas.

3.4. b) . Le pire cas devient celui où chaque acteur a déjà 4 ressources et où chacun en demande 3 nouvelles. Ce pire cas est servi avec 19 ressources et tous les acteurs pourront s'exécuter, l'un après l'autre, puisqu'aucun ne demande plus de 7 ressources et que chacun rend, au bout d'un temps fini, les ressources qu'il utilise. Avec 19 ressources, on n'a jamais d'interblocage

Question 3.5. a) si le code des acteurs change et s'ils peuvent demander les ressources une à une, on peut servir N acteurs sans interblocage si le nombre des ressources est 6*N + 1 = 15 d'où N = 2.

3.5. b). Pour N acteurs, le pire cas est celui où les N acteurs ont chacun 4 ressources (cela fait 4*N ressources) et où chacun demande 3 ressources de plus. Pour ne pas avoir d'interblocage, il suffit de pouvoir servir l'un d'entre eux, puisqu'alors, après exécution, on sait qu'il rendra toutes ses ressources. Donc 4*N + 3 >= 15, doù N = 3.

Question 3.6. limitation de concurrence

On doit créer une cohorte de 3 acteurs avec un sémaphore S initialisé à 3. Chaque acteur, avant de faire sa première demande de ressource, doit faire P(S) et après restitution des 7 ressources, il doit faire V(S).

```
package body Concurrence is
   -- on autorise au plus MaxProc processus
   SemConcurrence : Semaphore;  -- sert au contrôle
   procedure Entrer is
      begin P(SemConcurrence); end Entrer;
   procedure Sortir is
      begin V(SemConcurrence); end Sortir;
begin
   E0(SemConcurrence, MaxProc); -- initialiser les sémaphores utilisés
end Concurrence;
```

Question 3.7. 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
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 >= 7); -- un acteur au moins pourra obtenir son annonce
      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;
```

4. ARCHIVAGE DES DOCUMENTS

Solution Archivage

```
package body Rapport is
   T : array(0..9) of Document; Taille : Integer := 0; Tete, Queue : Integer := 0;
   -- déclarer les sémaphores nécessaires
   Mutex, Nplein, Nvide : Semaphore;
   procedure Deposer(X : in Document) is
   begin
      P(Nvide);
      P(Mutex);
      T((Queue) := X; Queue := (Queue + 1) mod 10;
      Taille := Taille + 1;
      V(Mutex);
      V(Nplein);
   end Deposer;
   procedure Retirer(Y : out TabDoc) is
      Nb : Integer; -- nombre de messages retirés
   begin
      -- attendre qu'il y ait au moins un message
      P(Nplein);
      P(Mutex);
      Y(1) := T(Tete); Tete := (Tete + 1) mod 10; Taille := Taille - 1;
      -- signaler la case vide
      V(Nvide);
      -- recopier au plus 5 autres messages s'il y en a
      if Taille < 5 then Nb := Taille; else Nb := 5; end if; -- noter que si Nb=0, alors 0 boucle
      for K in 1.. Nb loop
         -- mettre à jour le nombre des consommations restant à faire
         P(Nplein); -- grâce à Taille, on sait que ce n'est pas bloquant
         Y(K) := T((Tete + K -1) mod 10;
         -- signaler la case vide
         V(Nvide);
      end loop;
      tete := (Tete + Nb) mod 10;
      Taille := Taille - Nb; -- mettre à jour le nombre de messages restants
      V(Mutex);
   end Retirer;
begin
   -- initialiser les sémaphores utilisés
   E0(Mutex, 1); E0(Nplein 0); E0(Nvide, 10);
end Rapport;
```

5. CONTRÔLE DE CONCURRENCE PAR TÂCHE ADA SERVEUR

C'est du cours ou presque

L'allocution de ressources avec plusieurs ressources allouées FIFO est donné chapitre 10 Illustrations page 11

L'archivage des documents est du producteur consommateur simple à modifier

```
task body Rapport is
   T : array(0..9) of Document;
   Taille : Integer := 0; Tete, Queue : Integer := 0;
   Nb : Integer; -- nombre de messages pouvant être retirés
begin
   loop
      select
         when Taille < 10 => accept Deposer(X : in Document) do
            T(Queue) := X; end Deposer;
            Queue := (Queue + 1)mod 10; Taille := Taille + 1;
      or
         when Taille >0 => accept Retirer(Y : out TabDoc) do
            if Taille < 6 then Nb := Taille; else Nb := 6; end if;
            for K in 1.. Nb loop
                  Y(K) := T((Tete + K -1) mod 10);
               end loop;
            end Retirer;
            Taille := Taille - Nb; -- mettre à jour le nombre de messages restants
            Tete := (Tete + Nb) mod 10;
      or
         terminate;
      end select;
   end loop;
end Rapport;
```

La gestion de concurrence revient à programmer une cohorte. On introduit ici, pour montrer un aspect utile de Ada, une vérification du nombre de sorties. S'il y a plus de sorties que d'entrées, une exception est levée dans la partie rendez-vous ("do..end"), ce qui propage l'exception dans le serveur et aussi chez le client.

```
Task body Concurrence is
   EnCohorte : Integer := 0;
   Erreur : exception;
begin
   loop
      select
         when EnCohorte < maxProc =>
            accept Entrer; EnCohorte := EnCohorte + 1;
      or
         accept Sortir;
         do if EnCohorte = 0 then raise Erreur; end if; end Sortir
         EnCohorte := EnCohorte - 1;
      or
         terminate;
      end select;
   end loop;
exception
   when Erreur => Put("Mauvaise coordination car trop de sorties de la cohorte");
end Concurrence;
```