SYSTÈMES ET RÉSEAUX INFORMATIQUES

COURS B4 : HTO(19339) et ICPJ(21937)
CYCLE PROBATOIRE INFORMATIQUE
(Conception et développement informatique)

EXAMEN DU 16 JUIN 2001

portant sur l'enseignement des SYSTÈMES INFORMATIQUES

Date : samedi 16 juin 2001
Heure : 14h à 17h
Lieu : CNAM Amphi T

Texte de 5 pages plus 5 pages d'annexes
Les annexes sont également fournies sous forme de feuille préremplie pour la solution

TOUS DOCUMENTS AUTORISES

<<<<<<<<<<
L'examen comprend :
des questions sur la gestion des ressources
des questions sur la synchronisation

Chaque question peut-être traitée indépendamment.

Attention, vous n'avez pas de programme Ada à écrire (sauf la question 5), vous n'avez qu'à placer et initialiser des sémaphores et des variables globales.

Barème indicatif :
1. Unité centrale (5 points): 1.1 (1 point) + 1.2 (1 points) + 1.3 (1 point) + 1.4 (2 points)
2. Ordre de démarrage (3 points) : 2.1 (2 points)+ 2.2 (1 point)
3. Allocation des ressources (8 points) : 3.1. (1 point)+ 3.2. (2 points)+ 3.3. (1 point) + 3.4. (1 point)+ 3.5. (1 point)+ 3.6. (1 point)+ 3.7. (1 point)
4. Archivage : 3 points
5. Communication par message : 2 points

Ne pas écrire au crayon ni à l'encre rouge
(sous peine de nullité, selon le règlement des examens)

>>>

QUESTION 1. UTILISATION DE LA RESSOURCE UNITÉ CENTRALE

On a mesuré les durées d'exécution de 5 processus, T1, T2, T3, T4, T5, pour pouvoir étudier l'allocation de l'unité centrale. Pendant ces mesures les processus ne font pas d'entrées-sorties, mais uniquement du calcul. On connaît aussi les dates d'activation des processus. On a, pour chacun des 5 processus, les couples (date d'activation , durée d'exécution) suivants.
T1 = (0, 4) ; T2= (1, 4) ; T3 = (3, 2) ; T4 = (5, 4) ; T5 = (7, 2)
Par exemple T3 devient candidat à l'unité centrale à l'instant 3 et demande deux unités de temps d'exécution.

On note temps de réponse : TR = date de fin - date d'activation .

Question 1.1. Dans une première étude, on exécute les processus à l'ancienneté. Compléter la trace des exécutions donnée en annexe 1. Déterminer le temps de réponse de chaque processus ainsi que le temps de réponse moyen. L'annexe 1 fournit les tableaux à utiliser pour les calculs.

Question 1.2. On exécute maintenant les processus selon le tourniquet avec un quantum de 2. Compléter la trace des exécutions donnée en annexe 1. Déterminer le temps de réponse de chaque processus ainsi que le temps de réponse moyen. Comment expliquez-vous le résultat pour T1, T2 et T4 et le résultat moyen ?

Question 1.3. On suppose maintenant que T2 et T3 accèdent à une ressource critique en exclusion mutuelle après l'exécution d'une unité de temps hors section critique et qu'ils restent ensuite en section critique jusqu'à la fin de leur exécution.
On gère à nouveau l'unité centrale selon le tourniquet avec un quantum de 2. Compléter la trace des exécutions donnée en annexe 1. Déterminer le temps de réponse de chaque processus ainsi que le temps de réponse moyen.

Question 1.4. On veut favoriser les travaux courts comme T3 et T5, en leur attribuant une forte priorité (forte priorité = petite valeur) et en allouant l'unité centrale selon les priorités avec préemption. On donne les priorités suivantes :
5 à T1, 4 à T2, 1 à T3, 3 à T4, 2 à T5.
Compléter la trace des exécutions donnée en annexe 1. Déterminer le temps de réponse de chaque processus ainsi que le temps de réponse moyen. Quel phénomène explique la différence entre T5 et T3? Avec quel mécanisme pourrait-on l'éviter? Quel serait alors le temps de réponse de T3 ?

QUESTIONS 2 à 5

Système avec des processus concurrents

On étudie quelques aspects du système suivant qui comprend 6 processus concurrents : le processus Main, 4 processus Acteurs, un processus Gestionnaire, et des paquetages de contrôle de concurrence.

Chaque Acteur va exécuter un calcul compliqué, C, pour lequel il lui faut d'abord 2 ressources, puis 4 puis 7 ressources. A chaque cycle de calcul, l'Acteur produit un document qu'il envoie au gestionnaire G par le biais du paquetage Rapport.

Les processus sont concurrents et les 6 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.

Système avec des processus concurrents

procedure Main is

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

```
-- contrôle l'ordre de mise en route
package Depart is
   procedure Attendre(I : in Integer);
   procedure Signaler(J : in Integer);
end Depart;
```

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

```
-- 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'archivage des documents
package Rapport is
   procedure Deposer(X : in Document);
   procedure Retirer(Y : out TabDoc);
end Rapport;
```

```
-- déclaration, création du Gestionnaire

task G is
   Id : Integer := 1; -- nom unique
   LesRapports : TabDoc; -- 6 documents
begin

   -- attendre son tour pour imprimer
   Depart.Attendre(1);
   -- afficher son démarrage et son nom
   Put_Line(
      "Demarrage de " & Integer'Image(Id));
   -- annoncer qu'il a fini d'imprimer
   Depart.Signaler(1);

   loop
      Rapport.Retirer(LesRapports);
      Stocker(LesRapports) --  sur disque
   end loop;

end G;
```

```
-- déclaration du type Acteur

task type TypeActeur is
   -- nom unique du processus Acteur
   Id : Integer;
   --document établi par l'exécution de C
   MonCompteRendu : Document;
   begin

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

   -- attendre son tour pour imprimer
   Depart.Attendre(Id);
   -- afficher son démarrage et son nom
   Put_Line(
      "Demarrage de " & Integer'Image(Id));
   -- annoncer qu'il a fini d'imprimer
   Depart.Signaler(Id);

   Ressource.Allouer(1, Id);
   for K in 1..3 loop
      -- prendre K ressources de plus
      Ressource.Allouer(K, Id);
      C; --execution du code specifique
      --avec 2, puis 4 puis 7 ressources
      Rapport.Deposer(MonCompteRendu);
   end loop;
   -- rendre toutes les ressources
   Ressource.Restituer(7, 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 6 tâches démarrent
   -- afficher son démarrage et son nom
   Put_Line("Demarrage de Main" );
   --autoriser les autres annonces
   Depart.Signaler(0);
   Depart.Attendre(0); -- fin des annonces
   Put_line("Fin du demarrage");

end Main;
```

2. 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 6 processus démarrent en même temps.

Ordre 1. Les 6 processus ont des noms de 0 à 5. En effet, on a donné les noms 0 à Main et 1 au Gestionnaire et les 4 Acteurs prennent dynamiquement un nom unique, de 2 à 5.

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.

Question 2.1. Dans le paquetage Depart, on utilise un tableau SemBloc de 6 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 6) pour réveiller le processus (J + 1) mod 6. Compléter la programmation du paquetage Depart, donné en annexe 2, en utilisant SemBloc.

Ordre 2. 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. On ajoute une variable Comptage : Integer := 0; et on modifie la procedure Signaler, avec une instruction de choix multiple.

Question 2.2. Compléter la nouvelle programmation du paquetage Depart en utilisant SemBloc et Comptage, et en garantissant la cohérence de Comptage. Voir l'annexe 2.

3. ALLOCATION DES RESSOURCES

Synchronisation

a) On dispose de N ressources partagées entre les 4 acteurs. On gère l'allocation de ces ressources et l'attente éventuelle des demandeurs ainsi que la restitution des ressources et le réveil éventuel de demandeurs bloqués, dans le paquetage Ressource donné en annexe 3. Si on ne peut satisfaire le demandeur bloqué choisi, on arrête les réveils, sinon, s'il reste au moins un demandeur bloqué, on continue le service et on essaie de satisfaire le demandeur bloqué suivant choisi. On ne se préoccupe pas du contrôle de l'interblocage.

Question 3.1 Complétez le paquetage donné en annexe 3 avec les sémaphores de contrôle et en utilisant le schéma des sémaphores privés.

b) On veut, dès qu'une demande n'est pas satisfaite, cesser de prendre de nouvelles demandes.

Cela veut dire qu'on sert les demandes une à une, attente comprise. Que l'allocation se fasse sans blocage ou qu'elle se fasse avec blocage, on considère la procédure d'allocation comme une section critique, protégée par un sémaphore MutexService. Par contre, la restitution n'est pas concernée par cette exclusion mutuelle. Toutefois, Allouer et Restituer utilisent des variables d'état qui doivent rester cohérentes.

Désormais, il n'y a au plus qu'un processus bloqué par Allouer, ce qui simplifie le choix du candidat lors de restitution de ressources. Un seul des 4 sémaphores du tableau de sémaphores privés est utilisé pour bloquer ce processus. On peut les remplacer par un seul sémaphore SemCandidat.

Il vient alors, après simplifications, le paquetage donné en annexe 4.

Question 3.2. Complétez le paquetage donné en annexe 4 avec les sémaphores de contrôle et leur utilisation. Expliquez pourquoi il ne peut y avoir interblocage entre un processus qui utilise Allouer et un processus qui utilise Restituer.

Interblocage.
Chaque acteur demande successivement 1 ressource, puis 1 autre, puis 2 autres, puis 3 autres. Il obtient ainsi jusqu'à 7 ressources et il ne les restitue qu'à la fin. Quand il demande X ressources, et qu'elles ne sont pas disponibles, il n'en reçoit aucune et est bloqué.

Question 3.3. Si le système ne dispose que de 15 ressources, montrer qu'il peut y avoir interblocage.

Évitement. Pour que l'interblocage ne puisse se produire, il y a 3 méthodes : ajouter des ressources, réduire le nombre d'acteurs concurrents se partageant les ressources, mettre une prévention dynamique.

Question 3.4. Ajouter des ressources. Quel est le nombre minimum de ressources à ajouter ici quand :
a) On suppose que la suite des demandes peut être quelconque, sans dépasser 7 au total ?
b) On suppose que la suite des demandes sera toujours 1, + 1, + 2, + 3 ?

Question 3.5. Réduire le nombre d'acteurs concurrents se partageant les ressources. Avec 15 ressources, quel serait le nombre maximal d'acteurs concurrents à autoriser, dans les cas a) et b) indiqués ci-dessus ?

Question 3.6. On choisit de contrôler l'interblocage en limitant le nombre des acteurs qui peuvent demander concurrement des ressources. Pour cela, on introduit un paquetage de gestion de concurrence avec deux procédures Entrer et Sortir.

```
package Concurrence is
   procedure Entrer;
   procedure Sortir;
end Concurrence;
```

On ajoute dans la déclaration de task type TypeActeur les deux instructions :
Concurrence.Entrer; --après l'instruction Depart.Signaler()
Concurrence.Sortir; -- après l'instruction Ressource.Restituer()
Il vient alors :

```
package body Concurrence is
   -- on autorise au plus MaxProc processus
   SemConcurrence : Semaphore;  -- sert au contrôle
   procedure Entrer is
      begin -- contrôle de l'entrée de au plus MaxProc avec blocage éventuel; end Entrer;
   procedure Sortir is
      begin -- sortie annoncée, avec réveil éventuel d'un processus bloqué; end Sortir;
begin
   -- initialiser les sémaphores utilisés
end Concurrence;
```

Complétez le paquetage ci-dessus avec les sémaphores de contrôle et leur utilisation.

Question 3.7. Mettre une prévention dynamique. Quel algorithme utiliseriez vous? Décrivez le principe de cet algorithme. On pourra noter :
R : Integer; -- le résidu de ressources au moment de la demande
Total : array(2..5) of Integer; -- les allocations déjà faite aux 4 acteurs (de noms 2 à 5)

```
function EtatFiable(X : in Integer; I : in Integer) return Boolean; -- l'algorithme à décrire.
      -- les paramètres sont X : nombre de ressources demandées, I : nom du demandeur
```

4. ARCHIVAGE DES DOCUMENTS

Les acteurs déposent des documents et le gestionnaire retire un tableau qui contient de 1 à 6 documents, selon l'état du tampon. Aucun dépôt ne doit se faire pendant le retrait groupé.

Question 4. Complétez le paquetage donné en annexe 5, avec les sémaphores de contrôle et leur utilisation. L'exécution de Deposer doit être interdite pendant toute l'exécution de Retirer, que cela retire 1 seul ou jusqu'à 6 documents. Attention à ne pas créer d'interblocage par cette interdiction : quand Retirer est bloquant parce qu'il n'y a pas de document dans le tampon, Deposer doit pouvoir avoir accès au tampon T.

5. CONTRÔLE DE CONCURRENCE PAR TÂCHE ADA SERVEUR

On a vu dans le cours qu'on pouvait utiliser une tâche serveur en Ada, avec des points d'appel ("entry") et avec invocation à distance (par message) par les clients, pour implémenter un paquetage de contrôle.

Question 5. Choisir un des paquetages ci-dessous et donner la programmation de son corps ("body") avec une tâche serveur.

```
--remplacer le paquetage Ressource par
task Ressource is
   entry Allouer(X : in Integer; I : in Integer);
   entry Restituer(Y : in Integer; I : in Integer);
end Ressource ;

--remplacer le paquetage Rapport par
task Rapport is
   entry Deposer(X : in Document);
   entry Retirer(Y : out TabDoc);
end Rapport;

--remplacer le paquetage Concurrence par
task Concurrence is
   entry Entrer;
   entry Sortir;
end Concurrence ;
```

Examen de systèmes informatiques **numéro de copie :**
16 juin 2001

ANNEXE1
QUESTION 1: UTILISATION DE LA RESSOURCE UNITÉ CENTRALE

QUESTION 1.1 : Tableau à compléter pour la trace

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											
élu	T1	T1	T1	T1	T2											

QUESTION 1.2 : Tableau à compléter pour la trace

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											
élu	T1	T1	T2	T2	T1											

QUESTION 1.3 : Tableau à compléter pour la trace

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											
élu	T1	T1	T2	T2	T1											
S. critique				T2												

QUESTION 1.4 : Tableau à compléter pour la trace

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											
priorités	T1	T2	T2	T3	T2											
S. critique			T2		T2											

Unité centrale : Tableau à utiliser pour la 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		
T2	1	4	4								
T3	3	2	1								
T4	5	4	3								
T5	7	2	2								
mo yen ne											

ANNEXE 2
QUESTION 2 : ORDRE POUR LES ANNONCES DE DÉMARRAGE

Question 2.1

```
package body Depart is
   SemBloc : array(0..5) of semaphore;
   procedure Attendre(I : in Integer) is
   begin
      -- bloquer le processus de nom I

   end Attendre;

   procedure Signaler(J : in Integer) is
   begin
      -- réveiller le processus successeur de nom (J + 1) mod 6

   end Signaler;

begin
   -- initialiser chaque sémaphore du tableau SemBloc
   for K in 0..5 loop
       -- initialiser SemBloc(K);

   end loop;
end Depart;
```

Question 2.2.

```
package body Depart is
   SemBloc : array(0..5) of semaphore;
   Comptage : Integer := 0;

   procedure Signaler(J : in Integer) is
   begin
      case J is
         when 0 => -- réveiller le processus de nom 1;

         when 1 => -- réveiller les processus de noms 2, 3, 4 et 5;


         when 2..5 =>  -- quand J vaut une des valeurs de 2 à 5, on fait ce qui suit

                       Comptage := Comptage + 1;
                       if Comptage = 4 then      -- réveiller le processus de nom 0;

                        end if;

      end case
   end Signaler;
begin                -- initialiser chaque sémaphore utilisé

   for K in 0..5 loop
       -- initialiser SemBloc(K);

   end loop;
end Depart;
```

ANNEXE 3
QUESTION 3.1. : ALLOCATION DES RESSOURCES

```
package body Ressource is

   R : Integer := N; -- résidu, 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; -- chaque demande nouvelle est nulle initialement
   -- déclaration des sémaphores utilisés


   procedure Allouer(X : in Integer; I : in Integer) is
      -- demande de X ressources par le processus de nom I
      OK : Boolean; -- allocation possible ou non
   begin

      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

      else
         Attente := True; Attend(I) := True; Combien(I) := X;
         -- bloquer I en attente de ressource
      end if;


   end Allouer;

   procedure Restituer(Y : in Integer; I : in Integer) is
      -- restitution de Y ressources par le processus de nom I
      J : Integer;  -- nom d'un processus bloqué
      function ChoixCandidat return Integer; -- choisit un processus en attente
   begin

      R := R + Y; Total(I) := Total(I) - Y;
      While Attente loop
         J := ChoixCandidat;
         exit when R < Combien(J); -- pas assez de ressources pour J, on arrête les réveils
         Attend(J) := False; R := R - Combien(J); Combien(J) := 0; Attente := False;
         -- réveiller le processus J

         -- voir s'il y a au moins un autre processus en attente, pour continuer le service
         for K in 2..5 loop Attente := Attend(K); exit when Attente; end loop;
      end loop;

   end Restituer;

begin
-- initialisation des sémaphores utilisés


end Ressource;
```

ANNEXE 4
QUESTION 3.2 : ALLOCATION DES RESSOURCES (SUITE)

```
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, Attente = True si un processus est bloqué
   SemCandidat : Semaphore;
   MutexService : Semaphore;
   -- déclaration des autres sémaphores utilisés

   procedure Allouer(X : in Integer; I : in Integer) is
   begin
      -- entrée d'exclusion mutuelle entre les processus qui appellent Allouer

      -- garantir la cohérence des variables d'état utilisables aussi par la procedure Restituer

      R := R - X; --allocation des ressources à I, allocation immédiate ou prévue
      Total(I) := Total(I) + X;
      if R >= 0 then
         Attente := False;
         -- décider d'autoriser I à continuer son exécution

      else
         Attente := True;
         -- décider de bloquer I en attente de ressource en utilisant SemCandidat

      end if;
      -- cohérence des variables d'état utilisables aussi par la procedure Restituer

      -- blocage éventuel et

      -- sortie d'exclusion mutuelle entre les processus qui appellent Allouer


   end Allouer;

   procedure Restituer(Y : in Integer; I : in Integer) is
   begin

      R := R + Y; Total(I) := Total(I) - Y;
      If Attente and R >= 0 then          -- R >=0, on a rendu assez de ressources pour satisfaire
         Attente := False;                                    -- le processus bloqué
         -- réveiller le processus bloqué par SemCandidat

      end if;

   end Restituer;

begin
-- initialisation des sémaphores utilisés
```

```
end Ressource;
```

ANNEXE 5

ANNEXE 5 : ARCHIVAGE DES DOCUMENTS

```
package body Rapport is

   T : array(0..9) of Document; Taille : Integer := 0; Tete, Queue : Integer := 0;
   -- déclarer les sémaphores nécessaires



   procedure Deposer(X : in Document) is
   begin




   end Deposer;

   procedure Retirer(Y : out TabDoc) is
      Nb : Integer; -- nombre de documents retirés
   begin
      -- attendre qu'il y ait au moins un document

      -- garantir la cohérence des données

      Y(1) := T(Tete); Tete := (Tete + 1) mod 10; Taille := Taille - 1;
      -- signaler la case vide

      -- recopier au plus 5 autres documents s'il y en a
      if Taille < 5 then Nb := Taille; else Nb := 5; end if;  -- noter : si Nb=0, alors pas de boucle
      for K in 1.. Nb loop
         -- mettre à jour le contrôle du nombre des consommations restant à faire
         -- grâce à Taille,
         -- on sait que la primitive P, de contrôle des consommations, n'est pas bloquante

         Y(K + 1) := T(Tete + K -1) mod 10;
         -- signaler la case vide

      end loop;
      Tete := (Tete + Nb) mod 10;
      Taille := Taille - Nb; -- mettre à jour le nombre de documents restants

   end Retirer;

begin
   -- initialiser les sémaphores utilisés



end Rapport;
```