SYSTEMES ET RESEAUX INFORMATIQUES

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

EXAMEN DU 14 SEPTEMBRE 1999

partie portant sur l'enseignement des SYSTÈMES INFORMATIQUES

Date : mardi 14 septembre 1999
Durée : consacrer 1h30 sur les 3 heures de l'examen commun systèmes et réseaux informatiques
Heure : 18H15 - 21h15
Lieu : Salle 35-2-25 (Annexe rue Conté)

TOUS DOCUMENTS AUTORISES

L'examen comprend :
trois questions sur la synchronisation
deux questions sur l'optimisation des transferts disque

Chaque question peut-être traitée indépendamment.

Des annexes vous sont fournies pour vos réponses; ne pas oublier d'y reporter le numéro de votre copie (ne mettez ni votre nom, ni votre numéro de carte CNAM.

Attention, vous n'avez pas de programme Ada à écrire, vous n'avez qu'à placer et initialiser des sémaphores et des variables globales.

Barème indicatif de la partie systèmes :
Question 1 : 2pts
Question 2 : 2pts
Question 3 : 1pt
Question 4 : 3pts
Question 5 : 2pts

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

Problème

On construit, pour une agence multimédia, un serveur d'images qui comprend 50 processus transmetteurs et un disque géré par un processus disquaire.

Les 50 transmetteurs ont tous la même structure cyclique : chacun attend un appel externe pour traiter une requête de réception ou d'envoi d'une image ; pour pouvoir traiter une requête, le transmetteur utilise un élément de mémoire, appelé capsule, destiné à contenir l'image, le numéro de piste de stockage sur disque de l'image et le sens de la requête.

Lorsqu'il reçoit un appel externe, le transmetteur se fait allouer une capsule identifiée par un numéro ; s'il s'agit d'une réception d'image, il écrit l'image dans la capsule.

Dans tous les cas, il indique la piste de stockage de l'image, puis dépose le numéro de la capsule dans un tampon de communication avec le disquaire. Il attend un signal associé à la capsule et envoyé par le processus disquaire pour signifier la fin du transfert disque.

Si la requête est un envoi d'image, le transmetteur envoie l'image contenue dans la capsule.

Enfin le processus transmetteur restitue la capsule utilisée avant de se remettre en attente du prochain appel externe.

Le disquaire a aussi une structure cyclique : il attend 5 numéros de capsule dans le tampon de communication avec les processus transmetteurs ; il va utiliser ces 5 demandes pour optimiser les transferts du disque ; donc il lit dans chacune des 5 capsules la piste de disque de stockage de son image, effectue les transferts en jouant sur ces 5 pistes pour optimiser l'utilisation du disque (un transfert est une écriture si l'image est reçue, c'est une lecture si l'image doit être envoyée). Puis quand les transferts des 5 pistes sont terminés, le disquaire prévient les 5 transmetteurs concernés en envoyant à chacun un signal associé à sa capsule.

La structure de ces processus est donnée en fin d'énoncé (Compléments de programmation).

Pour régler la coopération entre ces processus, on met en place trois paquetages qui vont contenir les instructions de synchronisation. Ce sont les paquetages `Memoire`, `Tampon` et `Signalisation`.

Le paquetage `Memoire` gère 40 capsules ; un processus peut prendre un numéro de capsule ou en restituer un ; un processus peut lire et écrire dans les éléments d'une capsule de numéro donné ; ces éléments sont l'image contenue dans la capsule, le sens de la requête, la piste de stockage de l'image ; seules la prise et la restitution d'un numéro de capsule entraînent de la synchronisation, car 50 processus peuvent demander ces 40 capsules.

Le paquetage `Tampon` gère 20 cases pouvant contenir chacune comme message un numéro de capsule ; ces cases sont gérées à l'ancienneté ; les messages sont déposés un à un par les processus Transmetteurs alors que les messages sont lus par paquets de 5 à la fois.

Le paquetage `Signalisation` gère 40 signaux; chacun de ces signaux peut être attendu par un processus Transmetteur ou envoyé par le disquaire.

En associant un numéro de signal à un numéro de capsule, on peut imposer à un processus Transmetteur qui a reçu une capsule de numéro donné d'attendre jusqu'à ce que le contenu de cette capsule ait été traité par le disquaire.

Question 1 (2 points)

On considère ci-dessous une programmation des procédures `Prendre` et `Restituer` du paquetage Memoire.

On vous demande d'utiliser et d'initialiser les sémaphores `Mutex` et `S_Stock` pour prendre en compte la concurrence d'accès à ce paquetage. Utilisez pour votre réponse l'annexe 1 page 1.

```
with SRI_B; use SRI_B; with Table_Generique; use Table_Generique;
paquetage body Memoire is
  type D_A is (Disponible, Alloue);
  package Table is Table_Generique(Taille => 40, Element => D_A, Init =>
  Disponible);
    -- initialement toutes les capsules de Table sont disponibles
  Mutex, S_Stock : semaphore;
  procedure Prendre(A : out Positive) is
    Succes : boolean; K : Table.Index;
  begin
    Table.Rechercher(K, Disponible, Succes) ;
        --recherche une place K telle que Table(K) = Disponible
    Table.Changer(K, Alloue);
    A := K;
  end Prendre;
  procedure Restituer(B : in Positive) is
  begin
    Table.Changer(B, Disponible);
  end Restituer;
begin
  -- initialisation des sémaphores
end Memoire;
```

Question 2 (2 points)

On considère ci-dessous une programmation des procédures `Deposer` et `Retirer_5` du paquetage `Tampon`.

On vous demande d'utiliser et d'initialiser les sémaphores `Mutex_P`, `S_Plein` et `S_Vide` pour prendre en compte la concurrence d'accès à ce paquetage. Utilisez pour votre réponse l'annexe 1 page 1.

```
with SRI_B; use SRI_B;
paquetage body Tampon is
  Tamp : array (0.. 19) of Positive;
  Tete, Queue : Integer := 0;
  Mutex_P, S_Plein,S_Vide : semaphore;
  procedure Deposer(A : in Positive) is
  begin
    Tamp(Queue) := A;
    Queue := (Queue + 1) mod 20;
  end Deposer;
  procedure Retirer_5 ( B : out array(1..5) of Positive) is
  begin
    for I in 1..5 loop
      B(I) := Tamp(Tete);
      Tete := (Tete + 1) mod 20;
    end loop;
  end Retirer;
begin
  -- initialisation des sémaphores
end Tampon;
```

Question 3 (1 point)

On considère ci-dessous une programmation des procédures `Attendre` et `Envoyer` du paquetage `Signalisation.`

On vous demande d'utiliser et d'initialiser `S_Signal` le tableau de sémaphores pour prendre en compte la concurrence d'accès à ce paquetage.
Utilisez pour votre réponse l'annexe 1 page 2.

```
with SRI_B; use SRI_B;
paquetage Signalisation is
  S_Signal : array(1..40) of semaphore;

  procedure Attendre (C : in Positive) is
  begin
    -- attendre en utilisant le paramètre C
  end Attendre;

  procedure Envoyer(D : in Positive) is
  begin
    -- signaler en utilisant le paramètre D
  end Envoyer;
begin
  -- initialisation des sémaphores
end Signalisation
```

Question 4 (3 points)

Pour étudier l'optimisation des transferts disque obtenue en réordonnant les requêtes pour réduire les déplacements du bras, on trace une suite de 20 pistes qui sont fournies au disquaire et on compare trois résultats obtenus en faisant porter l'optimisation sur des paquets successifs de 5 requêtes, :
1) à l'ancienneté (fifo), c'est à dire en servant les requêtes selon leur arrivée,
2) le plus proche voisin de la position courante du bras (ppv),
3) l'ascenseur aller et retour (aar)

La position initiale du bras est à la piste 53, et pour l'ascenseur, le premier déplacement se fait selon les pistes croissantes.

La suite de pistes est la suivante, par paquets de 5 valeurs :

suite	98	83	37	22	14	24	65	67	36	02	34	09	12	17	18	04	11	16	43	91

Calculer le trajet total fait par la tête de lecture-écriture pour chacune des optimisations, en complétant les calculs suivants pour les 4 paquets de 5 valeurs de la suite:

fifo	98	83	37	22	14	24	65	67	36	02	34	09	12	17	18	04	11	16	43	91
trajet	45	15	46	15	8	10														

ppv	37	22	14	83	98	67														
trajet	16	15	8	69	15	31														

aar	83	98	37	22	14	02	24													
trajet	30	15	61	15	8	12	22													

*On constate que sur le premier paquet de 5 valeurs, le trajet total s'éléve à 129 pour fifo, à 123 pour **ppv** et 129 pour **aar**. L'optimisation semble peu efficace. Est ce pareil pour toute la trace?*
Utilisez pour votre réponse l'annexe 2 page 1.

Question 5 (2 points)

Pour améliorer l'efficacité de l'optimisation, on envisage de faire des paquets de 10 valeurs. Avant de modifier les programmes et d'écrire une procédure Tampon.Retirer_10, on veut connaître l'effet sur la suite de la question précédente.

La suite devient une suite de 2 paquets de 10 valeurs :

suite	98	83	37	22	14	24	65	67	36	02	34	09	12	17	18	04	11	16	43	91

Refaire les calculs pour les trois politiques fifo,ppv et aar par paquets de 10 sur la suite donnée. Quelles valeurs obtient-on pour le trajet total fait par la tête de lecture-écriture et quelles conclusions peut-on en tirer?
Utilisez pour votre réponse l'annexe 2 page 2.

--- COMPLEMENTS DE PROGRAMMATION (POUR INFORMATION) ---

Ces compléments ne sont donnés que pour information. On peut faire l'examen sans eux. Ne perdez donc pas de temps à comprendre ce code Ada si vous n'êtes pas familiers avec ce langage.

```
with SRI_B;use SRI_B;
procedure Exam_99_09 is -- PROGRAMME PRINCIPAL
   Type Sens is (Envoi, Reception); Type Image;

   package Tampon is    -- tampon de 20 cases contenant des entiers positifs, numéros de
   capsule
      procédure Deposer (A : in Positive); -- numéro de capsule
      procedure Retirer_5(B : out array(1..5) of Positive);  -- tableau de cinq numéros de
   capsule
   end Tampon;

   package Signalisation is     -- gère 40 signaux
      procédure Attendre (C: in Positive);  -- numéro de signal
      procedure Envoyer(D: in Positive);  -- numéro de signal
   end Signalisation;

   package Memoire is    -- gère 40 capsules
      -- une capsule comprend un sens, un numero de piste et une image
      procédure Prendre (A : out Positive);  -- A, B, C, E, G, I, K, M sont des numéros de
      capsule
      procedure Restituer(B : in Positive);  -- numéro de capsule
      procedure Ecrire_Sens (C : in Positive; D : in Sens); -- Type Sens is (Envoi,
   Reception)
      procedure Lire_Sens (E : in Positive; F : out Sens); -- Type Sens is (Envoi,
   Reception)
      procedure Ecrire_Piste (G : in Positive; H : in Positive); -- le numero de piste H
      varie de 1 a 100
      procedure Lire_Piste (I : in Positive; J : out Positive); -- le numero de piste J
      varie de 1 a 100
      procedure Ecrire_Image (K : in Positive; L : in Image); -- le type Image est donné
      procedure Lire_Image (M : in Positive; N: out Image); -- le type Image est donné
   end Memoire ;

   package body Tampon is separate;
   package body Signalisation is separate;
   package body Memoire is separate;
   task type Transmetteur; task type Disquaire;
   task body Transmetteur is
      T_Cap_Id : Positive; -- numéro de capsule
      T_Cap_Piste :Positive; -- numéro de piste de stockage sur disque
      T_Cap_sens : Sens; T_Cap_Image : Image;
   begin loop
      -- attendre un appel externe de reception ou d'envoi(T_Cap_Sens, T_Cap_Image)
      Memoire.Prendre(T_Cap_Id);
      -- obtenir une capsule s'il en reste, sinon attendre jusqu'à ce qu'il y en ait une
      de disponible
      If T_Cap_Sens = Reception then
      receptionner_l_image; Memoire.Ecrire_Image(T_Cap_Id , T_Cap_Image); end if;
      -- on suppose connue le numéro de piste T_Cap_Piste
      Memoire.Ecrire_Piste(T_Cap_Id, T_Cap_Piste); -- piste de disque à lire ou à écrire
      Tampon.Deposer(T_Cap_Id); -- on dépose le numéro de capsule
      Signalisation.Attendre(T_Cap_Id); -- attendre la fin du transfert de l'image
   contenue dans T_Cap_Id
      if T_Cap_Sens = Envoi then Memoire.Lire_Image(T_Cap_Image); envoyer_l_image; end if;
      Memoire.Restituer(T_Cap_Id);
   end loop; end Transmetteur;

   task body Disquaire is
      D_Cap_Id : array (1..5) of Positive; -- tableau de 5 numéros de capsule
      D_Cap_sens : array (1..5) of Sens; D_Cap_Image : array (1..5) of Image
```

```
      D_Cap_Piste : array (1..5) of Positive; -- tableau de 5 numéros de piste de stockage
      sur disque

   begin loop
      Tampon.Retirer_5(D_Cap_Id);
      for I in 1..5 loop
         Memoire.Lire_Piste(D_Cap_Id(I), D_Cap_Piste(I));
         Memoire.Lire_Sens(D_Cap_Id(I), D_Cap_Sens(I));
         if D_Cap_Sens(I) = Reception then
         Memoire.Lire_Image (D_Cap_Id(I); D_Cap_Image(I)); end if;
      end loop;
      Transfert_Disque(D_Cap_Piste, D_Cap_Image) ; -- concerne les 5 pistes et le
      transfert est optimisé
      -- selon le sens on lit ou on ecrit sur la piste Cap_Piste(I)
      for I in 1..5 loop
         Signalisation.Envoyer(D_Cap_Id(I));
      end loop;
   end loop; end Disquaire;
   Le_Transmetteur: array (1..50) of Transmetteur;
   Le_Disquaire: Disquaire;
begin null;
end Exam_99_06;

-------TABLE GENERIQUE : paquetage générique de parcours et modification d'une table
-------            construit et gère une table pouvant contenir jusqu'à Taille éléments de type Element
generic                                          Taille : Positive;         -- entier positif non nul
   type Element is <>;        -- element est un type énumératif et Element'First existe
Init : Element;
package Table_Generique is
   type Index is private;
   procedure Rechercher(I : out Index; B : in Element; Reussi : out boolean);
   procedure Changer(I : in Index; B : in Element);
private
   type Index is new Positive range 1..Taille;
end Table_Generique;
package body Table_Generique is
   Tab : array (Index) of Element := (others => Init); --initialise Tab avec Init
   procedure Rechercher(I : out Index; B : in Element; Reussi : out boolean) is
      J : Index := 1; -- on sait que Tab a au moins un élément
   begin
      while J < Taille loop
         exit when Tab(J) = B;
         J := J + 1;
      end loop;
      Reussi := Tab(J) = B;  -- Reussi est mis à vrai si Tab(J) = B
      -- il y a deux sorties possibles de la boucle : soit on a trouvé, soit on est arrivé sur le dernier
      -- soit on a : J< Taille et Tab(J) = B , soit on a : J = Taille et on n'a pas lu Tab(J)
      -- on doit reconnaître la sortie et tester Tab(J) si J = Tab'last
   end Rechercher;
   procedure Changer(I : in Index; B : in Element) is
      begin Tab(I) := B;
   end Changer;
   function Nombre(B : in Element) return Natural is -- indique le nombre de places de valeur B
      J : Index := 1; K : Integer := 0;
   begin
      while J < Taille loop
         if Tab(J) = B then K := K + 1; end if; J := J + 1;
      end loop;     -- il reste à examiner le contenu du dernier indice
      if Tab(J) = B then K := K + 1; end if;
      return K; end Nombre; end Table_Generique;                                -
```

ANNEXE 1 page 1 *Pour l'anonymat* ***, Mettez ici votre numéro de copie :***
Ne mettez ni nom, ni numéro de carte CNAM.

Question 1 (2pts)

```
with SRI_B; use SRI_B; with Table_Generique; use Table_Generique;
paquetage body Memoire is
    type D_A is (Disponible, Alloue);
    package Table is Table_Generique(Taille => 40, Element => D_A, Init =>    Disponible);
        -- initialement toutes les capsules de Table sont disponibles
    Mutex, S_Stock : semaphore;
    procedure Prendre(A : out Positive) is
        Succes : boolean; K : Table.Index;
    begin

        Table.Rechercher(K, Disponible, Succes) ;
                --recherche une place K telle que Table(K) = Disponible
        Table.Changer(K, Alloue);
        A := K;


    end Prendre;
    procedure Restituer(B : in Positive) is
    begin

        Table.Changer(B, Disponible);

    end Restituer;
begin
    -- initialisation des sémaphores


end Memoire;
```

Question 2 (2pts)

```
with SRI_B; use SRI_B;
paquetage body Tampon is
    Tamp : array (0.. 19) of Positive;
    Tete, Queue : Integer := 0;
    Mutex_P, S_Plein,S_Vide : semaphore;
    procedure Deposer(A : in Positive) is
    begin

        Tamp(Queue) := A;
        Queue := (Queue + 1) mod 20;

    end Deposer;
    procedure Retirer_5 ( B : out array(1..5) of Positive) is
    begin
        for I in 1..5 loop


            B(I) := Tamp(Tete);
            Tete := (Tete + 1) mod 20;


        end loop;

    end Retirer;
begin  -- initialisation des sémaphores

end Tampon;
```

ANNEXE 1 page 2 *Pour l'anonymat* ***, Mettez ici votre numéro de copie :***
Ne mettez ni nom, ni numéro de carte CNAM.
Question 3 (1 point)

```
with SRI_B; use SRI_B;
paquetage Signalisation is
    S_Signal : array(1..40) of semaphore;

    procedure Attendre (C : in Positive) is
    begin
        -- attendre en utilisant le paramètre C


    end Attendre;

    procedure Envoyer(D : in Positive) is
    begin
        -- signaler en utilisant le paramètre D


    end Envoyer;
begin
    -- initialisation des sémaphores


end Signalisation
```

ANNEXE 2 page 1 *Pour l'anonymat* ***, Mettez ici votre numéro de copie*** *:*
Ne mettez ni nom, ni numéro de carte CNAM.

Question 4 (3 pts)

Calculer le trajet total fait par la tête de lecture-écriture pour chacune des optimisations, en complétant les calculs suivants pour les 4 paquets de 5 valeurs de la suite:

fifo	98	83	37	22	14	24	65	67	36	02	34	09	12	17	18	04	11	16	43	91
trajet	45	15	46	15	8	10														

ppv	37	22	14	83	98	67														
trajet	16	15	8	69	15	31														

aar	83	98	37	22	14	02	24													
trajet	30	15	61	15	8	12	22													

On constate que sur le premier paquet de 5 valeurs, le trajet total s'éléve à 129 pour fifo, à 123 pour ***ppv*** *et 129 pour* ***aar****. L'optimisation semble peu efficace. Est ce pareil pour toute la trace?*

Question 5 (2pts)

La suite devient une suite de 2 paquets de 10 valeurs :

suite	98	83	37	22	14	24	65	67	36	02	34	09	12	17	18	04	11	16	43	91

Refaire les calculs pour les trois politiques fifo,ppv et aar par paquets de 10 sur la suite donnée. Quelles valeurs obtient-on pour le trajet total fait par la tête de lecture-écriture et quelles conclusions peut-on en tirer?

fifo																				
trajet																				

ppv																				
trajet																				

aar																				
trajet																				