SYSTÈMES ET RÉSEAUX INFORMATIQUES

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

EXAMEN DU MERCREDI 25 SEPTEMBRE 2002

portant sur l'enseignement des SYSTÈMES INFORMATIQUES

Date : mercredi 25 septembre 2002
Heure : 20h à 21h30

Lieu : CNAM

TOUS DOCUMENTS AUTORISES

(les ordinateurs portables ne sont pas autorisés)

****

Texte de 8 pages

L'examen comprend deux parties :
trois questions sur la synchronisation
trois questions sur la gestion des ressources

Chaque partie peut-être traitée indépendamment.

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

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

****

Barème indicatif (noté sur 10):

A. AGORA VIRTUELLE : 6 points

A1 (2 points) ;

A2 (2 points) ;

A3 (2 points) ;

B.GESTION DES RESSOURCES : 4 points

B1 : (1 point);

B2 : (2 points) ;

B3 : (1 point) ;

A. SYNCHRONISATION DE PROCESSUS
AGORA VIRTUELLE

Dans le commerce électronique (“e-commerce”), on utilise parfois une agora virtuelle (“virtual market”) comme lieu de rendez-vous entre clients qui veulent commercer en point à point en faisant du troc sans passer par un vendeur (“peer to peer exchange”). On met en place une telle agora et on la programme comme un paquetage.

Un processus client X appelle la procédure Agora.Rencontre(X, Y) pour obtenir le nom Y d’un associé.

Le paquetage Agora traite au maximum un nombre de 18 appels. Les autres sont mis en attente. Les appels reçus sont déposés dans un tampon interne et traités par trois serveurs. Chaque serveur prend deux appels, les associe et prévient les deux clients associés. La procédure Rencontre(X, Y) est réentrante et plusieurs clients peuvent l’appeler et l’exécuter en concurrence.

La procédure Rencontre(X, Y) comprend deux phases : le dépôt d’une requête avec le nom X puis l’attente d’un signal avant de récupérer le nom Y de l’associé et de sortir de l’agora. Un processus Serveur retire des requêtes deux à deux, note leur association dans une table d’association et envoie les signaux qui permettent aux deux clients concernés de sortir de l’agora. Il y a trois processus serveurs.

Les clients et les serveurs communiquent par un tampon T (qui est géré à l’ancienneté avec deux index Queue et Tete, et qui contient des noms de clients), une table d’association (nommée Complice) et par un tableau Sem de sémaphores qui servent à attendre et envoyer le signal de sortie de l’agora.

QUESTION A1.

On veut limiter à 18 le nombre de clients qui entrent dans l’agora. On contrôle cette cohorte de 18 clients avec un sémaphore appelé Seuil. Cela permet aussi de restreindre à 18 le nombre de sémaphores du tableau Sem et le nombre d’entrées dans la table Complice. Pour cela il faut donner à chaque processus un nom local IdLocal qui sert à utiliser ces sémaphores et la table Complice.

Fournir la programmation complète du paquetage body NomLocal (donné en partie dans l’annexe) pour que son utilisation par des processus concurrents reste cohérente.

Fournir la programmation complète de la procédure Rencontre(X, Y), donnée ci-après en annexe, pour assurer le contrôle de la cohorte tel que le nombre de clients dans l’agora ne dépasse jamais 18.

Les requêtes du tampon T contiennent un nom global X et un nom local I et elles sont du type Message. type Message is record X : Integer; I : IdLocal; end record;

QUESTION A2.

Chaque serveur prend deux requêtes une après l’autre, par deux appels à Requete.Retirer. Fournir la programmation complète de “package body Requete” avec les procédures Deposer et Retirer (schéma donné en annexe) . Ajouter les sémaphores nécessaires. (Ce qui importe ici c’est de bien synchroniser les processus pour qu’ils respectent la cohérence des données et des index du tampon). Programmer

de façon qu'un des serveurs et un des clients puissent avoir accès concurremment au tampon T, l'un en retrait, l'autre en dépôt (concurrence maximum pour l'accès au tampon T).

QUESTION A3.

Chaque serveur prend les deux requêtes par un appel unique à Requete.RetirerDeux.

a) Fournir la programmation complète de "package body Requete" donné en annexe (avec les procédures Deposer et RetirerDeux).Ajouter les sémaphores nécessaires. (Ce qui importe ici c'est de bien synchroniser les processus pour qu'ils respectent la cohérence des données et des index du tampon). Programmer Requete.RetirerDeux de façon qu'un des serveurs et un des clients puissent avoir accès concurremment au tampon T, l'un en retrait, l'autre en dépôt (concurrence maximum pour l'accès au tampon T).

b) Comparer qualitativement les deux solutions précédentes sur le plan du temps de réponse. Par exemple, supposons qu'il y ait trois requêtes dans le tampon (donc trois clients en attente de traitement par les serveurs). Montrer que, dans ce cas, une des solutions vues en A2 et A3 peut être moins bonne que l'autre. Donner le scénario correspondant.

ANNEXE DE PROGRAMMATION POUR LA QUESTION A

```
package body Agora is
    Max : constant := 18;       -- nombre maximum de clients dans l'agora
                                -- définition du type de nom local qui sert, dans Agora, à gérer les clients.
    type IdLocal is mod Max;
    type Message is         -- définition du type des messages
        record
            X : Integer;        -- nom global du client
            I : IdLocal;        -- nom local du client utilisé dans l'agora
        end record;
    -- ***                                  tampon de messages dans le paquetage Requete *****
    Package Requete is
        procedure Deposer(X : in Message);
        procedure Retirer(Y : out Message);
        procedure RetirerDeux(Y, Z : out Message);
    end Requete;
    --  ***
    package NomLocal is
                                -- pour la prise et restitution d'un nom local unique
        procedure Prendre(I : out IdLocal);
        procedure Rendre(I : in IdLocal);
    end NomLocal;
    --  ***
```

```
package body NomLocal is
    NumLibre : array(IdLocal) of Boolean := (others => True);
                                                    -- au début tous les noms sont libres
    << 1 :  à faire pour la question A1 >> : Semaphore; --  contrôle de cohérence
    procedure Prendre(I : out IdLocal) is
                -- On n'appelle cette procédure que lorsqu'on sait qu'il y a un nom inutilisé
    begin
        I := IdLocal'First; --  on utilise le paramètre I comme un index de parcours à partir de 0
        << 2 : à rendre cohérente  pour la question A1 >>
        loop
            exit when NumLibre(I);    -- on a trouvé un nom non utilisé, on sort de la boucle
            I := I + 1;
        end loop;
        NumLibre(I) := False;          -- on note qu'on utilise le nom I
        << 3 : à rendre cohérente  pour la question A1 >>
    end Prendre;
    procedure Rendre(I : in IdLocal) is
    begin
        << 4 : à rendre cohérente  pour la question A1 >>
        NumLibre(I) := True;
        << 5 : à rendre cohérente  pour la question A1 >>
    end Rendre;
begin                   --  *****    initialisations des sémaphores déclarés dans NomLocal**
<< 6 : à faire pour A1; mettre les valeurs initiales des sémaphores >>
end NomLocal;
--  ******************  *****************

    Sem : array(IdLocal) of Semaphore;     -- déclaration pour attente et envoi de signaux
    Complice : array(IdLocal) of Integer;  -- Complice(I) contient le nom global de l'associé de I
    --  ***
        --  contrôle de la cohorte pour que le nombre de clients dans l'agora ne dépasse jamais 18
    Seuil : Semaphore;          -- déclaré pour gérer la cohorte
    --  ***

    procedure Rencontre(X: in Integer; Y : out Integer) is
        I: IdLocal := 0; M : Message;
    begin
                                -- contrôle à l'entrée pour qu'il n'y ait que 18 clients dans l'agora
        << 7 : à faire  pour la question A1 >>
        NomLocal.Prendre(I);
                                -- envoi de la requête M
        M.X := X; M.I := I;
        Requete.Deposer(M); -- dépose sa demande
                                -- X attend le signal envoyé au nom local I pour sortir de la procédure
        P(Sem(I));
                                -- X lit le nom global de son associé
        Y := Complice(I);
                                -- X restitue le nom local I
        NomLocal.Rendre(I);
                            -- sortie contrôlée de l'agora pour assurer la présence de 18 clients au plus
        << 8: à faire  pour la question A1 >>
    end Rencontre;
--  ******************  *****************
```

```
task Serveur; task body Serveur is
    M, N : Message; A, B : Integer; J, K : IdLocal;
begin
    loop
        Requete.Retirer(M); Requete.Retirer(N);      -- il faut les requêtes de deux clients,
        -- Requete.RetirerDeux(A, B);    -- pour la question A3, remplace la ligne précédente
        A := M.X; B := N.X;                    -- récupérer les noms globaux
        J := M.I; K := N.I;                        -- récupérer les noms locaux
        Complice(J) := B; Complice(K) := A;       -- enregistrer les noms globaux des associés
        V(Sem(J)); V(Sem(K));                      -- envoyer les signaux à J et K
    end loop;
end Serveur ;
--  ***

Package body Requete is
                                                   -- déclarations pour gérer le tampon
    << à faire  pour chaque question A2 et A3 >> : Semaphore;
    type Index is mod 6;                           -- valeurs prises par les index du tampon T
    T : array(Index) of Message;
    Tete , Queue : Index := Index'First;           -- Index'First vaut 0 ici

    procedure Deposer(X : in Message) is
    begin
    << à faire  pour chaque question A2 et A3 >>
    end Deposer;

    procedure Retirer(Y : out Message) is
    begin
    << à faire  pour la question A2>>
    end Retirer;

    procedure RetirerDeux(Y, Z : out Message) is
    begin
    <<  à faire  pour la question A3 >>
    end RetirerDeux;

begin                  -- *****   initialisations des sémaphores déclarés dans Requete**
<< à faire pour A2 et A3 ; mettre les valeurs initiales des sémaphores >>
end Requete;
--  ******************** *****************


begin                  -- *****   initialisations des sémaphores déclarés dans Agora**
    << 9 : pour la question A1 ; mettre la valeur initiale de Seuil >>
    E0(Seuil, ); for I in IdLocal loop E0(Sem(I), 0); end loop;
end Agora;
-- ********************      fin de l'annexe AGORA      ********************
```

B. GESTION DES RESSOURCES

PRÉVENTION DE L'INTERBLOCAGE

Soit une application avec 4 processus et 8 ressources partagées. L'application a atteint l'état 1 ci-dessous où on note R le nombre des ressources non allouées. Pour chaque processus, on note son annonce (maximum du total de ressources qu'il a le droit de demander), les ressources qui lui sont allouées, la distance par rapport à l'annonce (soit, annonce - ressources allouées). On classe les processus selon la distance croissante.

On fait aussi apparaître les ressources récupérables par l'allocateur en récupérant les ressources allouées au processus de la colonne correspondante du tableau.

ÉTAT 1 R = 3 processus classés selon la distance croissante

Processus	P3	P4	P2	P1
Distance	2	3	5	6
Ressources allouées	1	1	2	1
Ressources récupérables : R initial = 3	4	5	7	8
Annonce	3	4	7	7

Question B1. Montrer que cet état est fiable, c'est à dire qu'en sérialisant leur exécution selon l'algorithme du banquier, l'allocateur pourrait servir tous les processus dans le cas extrême où ils demanderaient tous leur annonce.

B2. La file d'attente des requêtes de ressources demandées à l'allocateur contient des requêtes dans l'ordre suivant : (P1 demande 1 ressource en plus), (P2 demande 1 ressource en plus), (P4 demande 1 ressource en plus),(P3 demande 1 ressource en plus).

Une requête est acceptable, si une fois servie, elle fait passer le système dans un nouvel état fiable. Pour ce faire l'allocateur examine chaque requête avec l'algorithme du banquier.

Examen de la requête de P1 : ÉTAT 2.1 R = 2 avec la demande de P1

Processus	P3	P4	P2	P1
Distance	2	3	5	5
Ressources allouées	1	1	2	2
Ressources récupérables : R initial 2	3	4	6	8
Annonce	3	4	7	7

Examen de la requête de P2 : ÉTAT 2.2 R = 2 avec la demande de P2

Processus	P3	P4	P2	P1
Distance	2	3	4	6
Ressources allouées	1	1	3	1
Ressources récupérables : R initial 2	3	4	7	8
Annonce	3	4	7	7

Examen de la requête de P4 : ÉTAT 2.4 R = 2 avec la demande de P4

Processus	P3	P4	P2	P1
Distance	2	2	5	6
Ressources allouées	1	2	2	1
Ressources récupérables : R initial 2	3	5	7	8
Annonce	3	4	7	7

Examen de la requête de P3 : ÉTAT 2.3 R = 2 avec la demande de P3

Processus	P3	P4	P2	P1
Distance	1	3	5	6
Ressources allouées	2	1	2	1
Ressources récupérables : R initial 2	4	5	7	8
Annonce	3	4	7	7

Question B2. Indiquer toutes les requêtes qui sont acceptables et dites pourquoi le nouvel état est ou non un état fiable.

B3. L'allocateur choisit la première requête acceptable de la file d'attente, ce qui fait passer l'application à l'état 2 (dans cet état 2, on a R = 2).

Question B3.a. Quelles sont, parmi les trois requêtes restantes, celles qui seraient acceptables à partir de cet état 2 ? Dites pourquoi le nouvel état est ou non un état fiable.

L'allocateur choisit à nouveau la première requête acceptable parmi les deux requêtes restantes de la file d'attente, ce qui fait passer l'application à l'état 3 (dans cet état 3, on a R = 1).

Question B3.b. L'une des deux requêtes restantes est-elle acceptable à partir de l'état 3 ? Justifiez votre réponse.