**SYSTÈMES ET RÉSEAUX INFORMATIQUES**

**COURS B4 : HTO(19339) et ICPJ(21937)**

**CYCLE PROBATOIRE INFORMATIQUE**

**(Conception et développement informatique)**

# EXAMEN DU 16 SEPTEMBRE 2003

**portant sur l'enseignement des SYSTÈMES INFORMATIQUES**

**Date** : mardi 16 septembre 2003
**Heure** : 18h à 19h30

**Lieu** : CNAM salle 11.A2.31

**TOUS DOCUMENTS PAPIERS AUTORISES**

**(les ordinateurs portables ne sont pas autorisés)**

---

Texte d'énoncé de 4 pages
L'examen portant sur les systèmes informatiques comprend :
des questions sur la synchronisation des processus
des questions sur la gestion des ressources

## Chaque question peut-être traitée indépendamment.

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

---

**Barème *indicatif* pour la partie systèmes:**

EXERCICE SUR LA SYNCHRONISATION DES PROCESSUS⊡(2 + 3 + 2 = 7 points)
EXERCICE SUR LES RESSOURCES⊡(5 points)

---

L'examen de septembre pour le cours B4 comprend deux parties de 1h30 chacune, une en réseaux et une en systèmes. Chaque partie compte pour 50% de la note finale

## EXERCICE SUR LA SYNCHRONISATION DES PROCESSUS

Le service d'accès à la banque B comporte N processus serveurs prenant chacun en charge un client. Chaque serveur définit une requête qui contient une priorité, l'identité du serveur et un booléen indiquant si la requête consulte et/ou modifie le fichier des clients. Le fichier des clients est une ressource partagée entre ces serveurs.

On veut gérer cette ressource partagée de telle façon qu'en cas d'attente les processus soient servis, non pas à l'ancienneté, mais selon les priorités attribuées aux processus serveurs.

Cette gestion utilise les types suivants ainsi que le paquetage FileAttente qui est fourni :

```
subtype ProcessId is Integer range 1..N;
subtype Priority is Integer range 1..N;
type Triplet is record
    Id: ProcessId; -- identité du demandeur
    Prio: Priority; -- priorité du demandeur
    Modification: Boolean;
    -- indique si l'accès modifie ou consulte seulement le fichier
end record;
```

```
package FileAttente is
    -- gestion de file de Triplets triés par priorité décroissante
    -- la file peut contenir jusqu'à N Triplets
    -- sa gestion utilise des données communes et persistantes
    -- partagées entre les diverses procédures et les serveurs
    procedure Inserer(T: in Triplet); -- ajoute T dans la file
    function Premier return Triplet; -- lit le plus prioritaire
    procedure OterPremier; -- supprime le plus prioritaire
    function EstVide return Boolean; --vrai si la file est vide
end FileAttente;
package body FileAttente is begin …
    -- ** ce paquetage est fourni, vous n'avez pas à le programmer
end FileAttente;
```

Le problème consiste à étudier diverses formes pour l'implantation du corps du paquetage `Ressource` dont la spécification est donnée ci-après :

```
package Ressource is
    procedure Reserver(PourMoi: in ProcessId;
                       MaPrio: in Priority; Modif: in Boolean);
    procedure Liberer;
end Ressource;
```

## S1. PREMIÈRE SOLUTION AVEC EXCLUSION MUTUELLE

Tout accès à la ressource, quel qu'il soit, se fait en exclusion mutuelle entre les processus serveurs. La procédure `Reserver` assure le respect de cette exclusion mutuelle et classe les processus bloqués selon leur priorité. S'il y a un processus bloqué au moment de la libération de la ressource, la procédure `Liberer` réveille le processus bloqué le plus prioritaire.

On vous demande d'appliquer le schéma des sémaphores privés pour compléter le corps du paquetage `Ressource` ci-après et pour permettre une utilisation concurrente des procédures `Reserver` et `Liberer`. Toute utilisation de la ressource se fait alors par

`Reserver` (…) ; AccèsALaRessource ; `Liberer` ;

Le corps du paquetage `Ressource` à compléter est le suivant :

```
package body Ressource is
    -- données partagées entre tous les serveurs
    AccesExclusif: Boolean := False;
    Sempriv: array(ProcessId) of Semaphore;
    Mutex: semaphore;
```

```
    procedure Reserver(PourMoi□: in ProcessId□;
                       MaPrio□: in Priority□; Modif□: in Boolean)□is
        T□: Triplet□; Succes□: Boolean□;
    begin
    --**l'accès aux données partagées doit rester cohérent(à faire)
        -- premier cas, la ressource est occupée,
        -- on construit un triplet à placer dans la file d'attente
        if AccesExclusif then
            Succes□:= False□;
            T.Id□:= PourMoi□;
            T.Prio□:= MaPrio□;
            T.Modification□:= Modif;
            FileAttente.Inserer(T)□; -- insère selon la priorité
        -- deuxième cas, la ressource est libre
        else
            Succes□:= True□;
            AccesExclusif□:= True□;
        end if□;
    --**Ne pas oublier de bloquer l'appelant si Succes est faux□;
    end Reserver□;

    procedure Liberer□is
        T□: Triplet□;
    begin
    --**l'accès aux données partagées doit rester cohérent(à faire)
        if FileAttente.EstVide then
            AccesExclusif:= False; -- pas de processus en attente
        else
            T□:= FileAttente.Premier□; FileAttente.OterPremier□;
            -- a extrait le plus prioritaire de la file
            --**Réveiller le processus désigné par T.Id□; (à faire)
        end if□;
    end Liberer;

begin -- ** initialiser les sémaphores utilisés (à faire)
end Ressource□;
--** on a noté --** les commentaires qui guident votre travail
```

## S2. DEUXIÈME SOLUTION AVEC EXCLUSION MUTUELLE

On utilise un processus supplémentaire, `Gerant`, pour contrôler l'utilisation de la ressource en exclusion mutuelle. Quand un processus serveur utilise la procedure `Reserver` , il fait appel à ce processus gérant via un schéma producteurs-consommateur implanté par le paquetage `Requete.` Les serveurs produisent des requêtes et le gérant les consomme. Après avoir produit une requête, chaque serveur attend le feu vert du gérant avant d'utiliser la ressource. La gestion de l'exclusion mutuelle n'est plus faite par les serveurs, mais uniquement par le gérant, qui fait aussi partie du nouveau paquetage `Ressource.` Ce gérant autorise les serveurs à utiliser la ressource l'un après l'autre selon la priorité des demandes. Quand le gérant a réveillé un serveur en attente, il se met lui-même en attente de la sortie de section critique de ce serveur. Quand le serveur libère la ressource, il réveille alors le gérant pour que celui-ci puisse l'attribuer à un autre demandeur.

On vous demande de compléter, avec des sémaphores, le corps du paquetage `Requete` et celui du nouveau paquetage `Ressource` ci-après pour permettre une utilisation concurrente des procédures `Deposer` et `Retirer` de même que `Reserver` et `Liberer`.

```
Package Requete is
    procedure Deposer(X□: in Triplet)□;
    procedure Retirer(Y□: out Triplet)□;
end Requete□;
```

```
Package body Requete is -- utilise le paquetage FileAttente
    -- ** déclarer les sémaphores nécessaires à ce paradigme
    -- ** et compléter avec les opérations sur ces sémaphores
    -- si on veut la file d'attente peut contenir jusqu'à N requêtes
    procedure Deposer(X□: in Triplet)□is
    begin
        --** opérations de contrôle de concurrence et de cohérence
        FileAttente.Inserer(X)□; -- dépôt de X
        --** opérations de contrôle de concurrence et de cohérence
    end Deposer□;
    procedure Retirer(Y□: out Triplet)□is
    begin
        --** opérations de contrôle de concurrence et de cohérence
        Y□:= FileAttente.Premier□; FileAttente.OterPremier□;
        -- retrait du triplet le plus prioritaire de la file
        --** opérations de contrôle de concurrence et de cohérence
    end Retirer□;
begin --** initialiser les sémaphores utilisés
end Requete□;

package body Ressource is
    Signal□: array(ProcessId) of Semaphore□; -- un par serveur
    SortieAcces□: Semaphore□;
    -- avec Signal(I), on bloque et réveille le serveur I,
    -- avec SortieAcces, on bloque et réveille le gérant

    procedure Reserver(PourMoi□: in ProcessId□;
                       MaPrio□: in Priority□; Modif□: in Boolean)□is
        T□: Triplet□;
    begin
        T.Id□:= PourMoi□;
        T.Prio□:= MaPrio□;
        T.Modification□:= Modif;
        Requete.Deposer(T)□;
        -- ** bloquer l'appelant PourMoi en attente d'un Signal qui
        -- l'autorisera à accéder au fichier en exclusion mutuelle
    end Reserver□;

    procedure Liberer□is
    begin
        -- ** réveiller le gérant bloqué en attente de cette sortie
    end Liberer;

    task Gerant□;
    task body Gerant is
        T□: Triplet□; PourQui□: ProcessId□;
    begin
        loop
            -- consomme une requête
          Requete.Retirer(T)□;-- T le plus prioritaire de la file
          PourQui□:= T.Id□;
          --** Envoyer le signal de réveil au serveur PourQui
          --**Se bloquer en attente de sa libération du fichier
        end loop
    end Gerant□;

begin --**Initialiser les sémaphores utilisés par Reserver,
      -- par Liberer et par la tâche Gerant
end Ressource□;
```

## S3. TROISIÈME SOLUTION AVEC LECTEURS ET RÉDACTEURS

Comme dans la deuxième solution, le processus `Gerant`, retire les requêtes classées selon leur priorité. Si des requêtes successives ne font que des consultations (`Modification`= `False`), alors le gérant les autorise à accéder ensemble à la ressource (situation de lecteurs). Si une requête demande une modification (situation de rédacteur), le gérant attend la fin de toutes les requêtes de lecture en cours avant d'autoriser l'accès avec modification pour un seul serveur. Toute utilisation de la ressource se fait toujours par

`Reserver (…); AccèsALaRessource; Liberer;`

On vous demande de compléter avec des sémaphores la programmation du gérant dans sa nouvelle version suivante :

```
task Gerant;
task body Gerant is
    T: Triplet; PourQui: ProcessId;
    AccesExclusif: Boolean:= False; --accès en écriture
    NbConsulte: Integer:= 0 -- nombre d'accès en lecture
    -- ces variables sont locales au gérant, et non partagées
begin
    loop
      -- consomme une requête
      Requete.Retirer(T);-- T le plus prioritaire de la file
      PourQui:= T.Id;
      --cas où la ressource est occupée par un rédacteur
      if AccesExclusif then
        --** Attendre la fin des accès du rédacteur
        AccesExclusif:= False;
      end if;
      -- la ressource est libre ou en accès lecture seule
      -- c'est un nouvel accès en lecture, on l'autorise
      if not T.Modification then
        NbConsulte:= NbConsulte + 1;
        -- on va autoriser la lecture concurrente
      else
        -- c'est un accès en écriture, on doit
        -- attendre la fin de toutes les lectures en cours
        while NbConsulte > 0 loop
            --** attendre la fin d'un accès (il est en lecture)
            NbConsulte:= NbConsulte - 1;
        end loop;
        -- maintenant on va pouvoir autoriser l'écriture
        AccesExclusif:= True;
      end if;
      -- ** envoyer le signal de réveil au serveur PourQui
    end loop
end Gerant;
```

## EXERCICE SUR LES RESSOURCES

R1. Un programme parcourt une image qui occupe 8 pages de mémoire pour y rechercher un premier motif, puis un second, puis un troisième. Ce programme fait ainsi la suite des références (aux pages) suivante□

1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8

On suppose qu'il s'exécute dans un système paginé à la demande et qu'on lui a attribué une partition fixe de 6 cases.

a) Sans calculs, mais par un simple raisonnement, montrer qu'il n'y a pas de différence entre la politique de remplacement à l'ancienneté («FIFO») et la politique de remplacement de la page la moins récemment utilisée («LRU»). Donner alors le nombre de défauts de page.

On rappelle que, lors du démarrage du programme, toutes les cases sont initialement vides. Par conséquent toutes les premières pages coûtent chacune un défaut de page.

b) Sans calculs, donner le nombre de défauts de pages quand il s'exécute dans une partition fixe de trois pages.

Pour la suite, on suppose que le système implante toujours la politique de remplacement à l'ancienneté («FIFO»).

R2. On suppose qu'un motif ne peut être à cheval sur les deux moitiés de l'image. Le programme peut alors parcourir chaque moitié d'image successivement. Cela donne la suite de références suivante□

1 2 3 4 1 2 3 4 1 2 3 4 5 6 7 8 5 6 7 8 5 6 7 8

a) Donner le nombre de défauts de pages quand il s'exécute dans une partition fixe de quatre pages.

b) Donner le nombre de défauts de pages quand il s'exécute dans une partition fixe de trois pages.

R3. On change le sens de parcours à chaque fois. Cela donne la suite de références suivante□

1 2 3 4 4 3 2 1 1 2 3 4 5 6 7 8 8 7 6 5 5 6 7 8

a) Donner le nombre de défauts de pages quand il s'exécute dans une partition fixe de quatre pages.

b) Donner le nombre de défauts de pages quand il s'exécute dans une partition fixe de trois pages.

c) Qu'aurait donné (en nombre de défauts de page) la politique de remplacement LRU avec une partition fixe de trois pages.

SYNCHRONISATION DES PROCESSUS□PREMIÈRE SOLUTION

```
package body Ressource is
    -- données partagées entre tous les serveurs
    AccesExclusif: Boolean□:= False;
    Sempriv□: array(ProcessId) of Semaphore□;
    Mutex□: semaphore□;

    procedure Reserver(PourMoi□: in ProcessId□;
                       MaPrio□: in Priority□; Modif□: in Boolean)□is
        T□: Triplet□; Succes□: Boolean□;
    begin

        -- premier cas, la ressource est occupée,
        -- on construit un triplet à placer dans la file d'attente
        if AccesExclusif then
            Succes□:= False□;
            T.Id□:= PourMoi□;
            T.Prio□:= MaPrio□;
            T.Modification□:= Modif;
            FileAttente.Inserer(T)□; -- insère selon la priorité

        -- deuxième cas, la ressource est libre
        else
            Succes□:= True□;
            AccesExclusif□:= True□;

        end if□;


    end Reserver□;

    procedure Liberer□is
        T□: Triplet□;
    begin

        if FileAttente.EstVide then
            AccesExclusif:= False; -- pas de processus en attente

        else
            T□:= FileAttente.Premier□; FileAttente.OterPremier□;
            -- a extrait le plus prioritaire de la file


        end if□;

    end Liberer;

begin


end Ressource□;
```

```
Package body Requete is -- utilise le paquetage FileAttente


    procedure Deposer(X : in Triplet) is
    begin


        FileAttente.Inserer(X); -- dépôt de X


    end Deposer;
    procedure Retirer(Y : out Triplet) is
    begin


        Y := FileAttente.Premier; FileAttente.OterPremier;


    end Retirer;
begin

end Requete;

package body Ressource is
    Signal : array(ProcessId) of Semaphore;
    SortieAcces : Semaphore;

    procedure Reserver(PourMoi : in ProcessId;
                       MaPrio : in Priority; Modif : in Boolean) is
        T : Triplet;
    begin
        T.Id := PourMoi; T.Prio := MaPrio;
        T.Modification := Modif;
        Requete.Deposer(T);

    end Reserver;

    procedure Liberer is
    begin

    end Liberer;

    task Gerant;
    task body Gerant is
        T : Triplet; PourQui : ProcessId;
    begin
        loop
          Requete.Retirer(T);-- T le plus prioritaire de la file
          PourQui := T.Id;


        end loop
    end Gerant;

begin

end Ressource;
```

SYNCHRONISATION DES PROCESSUS□TROISIÈME SOLUTION

```
task Gerant;
task body Gerant is
    T: Triplet; PourQui: ProcessId;
    AccesExclusif: Boolean:= False; --accès en écriture
    NbConsulte: Integer:= 0-- nombre d'accès en lecture
    -- ces variables sont locales au gérant, et non partagées
begin
    loop
      -- consomme une requête
      Requete.Retirer(T);-- T le plus prioritaire de la file
      PourQui:= T.Id;
      --cas où la ressource est occupée par un rédacteur
      if AccesExclusif then


        AccesExclusif:= False;
      end if;
      -- la ressource est libre ou en accès lecture seule
      -- c'est un nouvel accès en lecture, on l'autorise
      if not T.Modification then
        NbConsulte:= NbConsulte + 1;
        -- on va autoriser la lecture concurrente
      else
        -- c'est un accès en écriture, on doit
        -- attendre la fin de toutes les lectures en cours
        while NbConsulte > 0 loop


            NbConsulte:= NbConsulte - 1;
        end loop;
        -- maintenant on va pouvoir autoriser l'écriture
        AccesExclusif:= True;
      end if;


    end loop
end Gerant;
```