SYSTEMES ET RESEAUX INFORMATIQUES

COURS B4 : HTO(19339) ICPJ(21937)
CYCLE PROBATOIRE INFORMATIQUE
(Conception et Développement Informatique)

EXAMEN DU 22 JUIN 1999

partie portant sur l'enseignement des systèmes informatiques

Date : mardi 22 juin 1999
Durée : 3 heures
Heure : 18h30 21h30
Lieu : Lariboisière

TOUS DOCUMENTS AUTORISES

L'examen comprend :
deux parties synchronisation (**Entrepôt, Transaction**)
une partie gestion de mémoire (**Va et vient entre Stock1 et Stock2**)
une partie synchronisation facultative(**Démon de minuit**).

Chaque partie 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 :
Question 1 : 5 pts
Question 2 : 5 pts
Question 3 : 3 pts
Question 4 : 4 pts
Question 5 : 3 pts
Question 6 : 3 pts

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

Problème

A la demande d'une entreprise qui utilise des documents informatisés, on étudie la mise en place d'un entrepôt de documents et de transactions utilisant cet entrepôt.

Entrepôt

Le cahier des charges stipule que l'entrepôt contient des documents accessibles par une clé fournie par l'entrepôt lors du premier dépôt d'un document, qu'un document doit pouvoir être consulté, qu'un document doit pouvoir être modifié en le remplaçant par une nouvelle version et enfin qu'un document doit pouvoir être supprimé. On exprime cela par la spécification de paquetage suivante :

```
------------------------------------ ENTREPOT
package Entrepot is
   type Cle is Positive; -- la clé est fournie par l'entrepôt
   type Document;     --  type défini ailleurs, sans utilité ici
   procedure Déposer (D : in Document; Ma_Cle : out Cle);
      -- dépot de D et réception de Ma_cle
   procedure Consulter (D : out Document; Ma_Cle : in Cle);
      -- avec Ma_Cle, réception de D
   procedure Modifier (D : in Document; Ma_Cle : in Cle);
      -- nouvelle version associée à Ma_Cle
   procedure Supprimer (Ma_Cle : in Cle);
      -- le document est détruit, Ma_Cle ne fournira plus rien
end Entrepot;
```

On veut implémenter le paquetage `Entrepot` en utilisant deux tableaux de documents :

- `Stock1`, qui est de petite taille (50 documents) et d'accès rapide, et que l'on remplit en premier

- `Stock2`, qui est de très grande taille (5000 documents) et d'accès plus lent, et que l'on remplit quand il n'y a plus de place dans `Stock1`.

On utilise aussi deux tables d'occupation `Table1` et `Table2` qui nous renseignent sur le remplissage des deux tableaux `Stock1` et `Stock2`, et qui contiennent les clés des documents stockés :

Une place libre à l'emplacement `J` de `Stock1` se représente par un `0` dans `Table1(J)`. Un document déposé dans `Stock1(K)` est représenté par sa clé dans `Table1(K)`. Il en est de même pour `Stock2` et `Table2`. Toute recherche d'un document, connu par sa clé `Ma_Cle`, commence par la recherche de cette clé dans la table d'occupation `Table1`, puis dans la table d'occupation `Table2` si on ne l'a pas trouvée dans `Table1`.

Les procédures de ce paquetage doivent être utilisées concurremment par des processus.

La programmation du paquetage `Entrepot` a été faite en sous-traitance en supposant que toutes les procédures étaient exécutées en exclusion mutuelle entre elles (`Entrepot` constitue une seule ressource critique protégée par un sémaphore `Mutex`) et en supposant que `Stock2` était de taille infinie. On a utilisé un paquetage générique qui sert de modèle pour créer une table d'occupation, y rechercher un élément (un 0 ou une clé) et y écrire un élément (une clé ou un 0). La programmation de ce paquetage est donnée en fin d'énoncé pour information.

La programmation du paquetage `Entrepot` fournie est la suivante :

```
with SRI_B; use SRI_B; with Table_Generique; use Table_Generique;
   -- SRI_B importe les sémaphores et les opérations P, V et E0
```

```
package body Entrepot is

   N1 : constant Positive := 50; N2 : constant Positive := 5000;
   N : constant := N1 + N2;
      -- total des places dans l'entrepôt
   subtype Index1 is Positive range 1..N1;
   subtype Index2 is Positive range 1..N2;
   -- données communes partagée par les procédures du paquetage
   Stock1 : array (Index1) of Document;
   Stock2 : array (Index2) of Document;
   package Table1 is new Table_Generique (taille => N1, Element => Natural);
   package Table2 is new Table_Generique (taille => N2, Element => Natural);
   Numero : Natural := 0;
      -- sert a numeroter les clés de façon unique jusqu' à Integer'Last
   Mutex_Entrepot : Semaphore;


procedure Deposer (D : in Document; Ma_Cle : out Cle) is
   K1 : Table1.Index; K2 : Table2.Index; Succes : boolean;
begin
   P(Mutex_Entrepot);
   Numero := Numero + 1; Ma_Cle := Numero;
   -- Ici on suppose qu'il y a toujours assez de place dans Table1 ou Table2 et  -- Stock1
ou Stock2 car on suppose que Stock2 est de taille infinie (N2 = infini   -- pour
l'application!!) attention, cette hypothèse n'est pas contrôlée par ce   -- programme en
l'état actuel. Pour une autre implémentation, il faudra en tenir   -- compte et ajouter un
contrôle
   Table1.Rechercher(K1, 0, Succes);
   -- recherche une place K1 telle que  Table1(K1) = 0
   if Succes then
   -- la recherche est  réussie
      Table1.Changer(K1, Ma_Cle);
      -- on note l'occupation dans Table1 avec Ma_Cle
      Stock1(K1) := D;
      -- on écrit dans stock1 et c'est fini
      V(Mutex_Entrepot);
      return;                                -- sortie de la procedure Deposer
   end if;
   -- on n'a pas trouvé dans Table1, on recherche dans Table2 qui est considérée de    --
taille infinie
   Table2.Rechercher(K2, 0, Succes);
   -- recherche une place K2 telle que Table2(K2) = 0 ;
   Table2.Changer(K2, Ma_Cle );
   -- on est sûr de trouver une place dans Table2
   Stock2(K2) := D;
   -- on écrit dans Stock2 et c'est fini
   V(Mutex_Entrepot);
end Deposer ;


procedure Consulter (D : out Document; Ma_Cle : in Cle) is
   K1 : Table1.Index; K2 : Table2.Index; Succes : boolean;
begin
   P(Mutex_Entrepot);
   Table1.Rechercher(K1, Ma_Cle, Succes);
   -- recherche K1 tel que Table1(K1) = Ma_Cle
   if Succes then
   -- on a trouvé, la recherche est Succes
      D := Stock1(K1);
      -- on lit dans stock1 et c'est fini
      V(Mutex_Entrepot);
      return;
```

```
      -- sortie de la procedure Consulter
   end if;
   -- on n'a pas trouvé dans Table1, on cherche dans Table2
   Table2.Rechercher(K2, Ma_Cle, Succes);
      -- on est sûr de trouver Table2(K2) = Ma_Cle
   D := Stock2(K2);
      -- on lit dans stock2 et c'est fini
   V(Mutex_Entrepot);
end Consulter;


procedure Supprimer (Ma_Cle : in Cle) is
   K1 : Table1.Index; K2 : Table2.Index; Succes : boolean;
begin
   P(Mutex_Entrepot);
   Table1.Rechercher(K1, Ma_Cle, Succes);
   -- recherche K1 tel que Table1(K1) = Ma_Cle
   if Succes then
   -- la recherche est un Succes
      Table1.Changer(K1, 0);
      -- on libère la place en inscrivant 0
      V(Mutex_Entrepot);
      return;
      -- sortie de la procedure Supprimer
   end if;
   --  on n'a pas trouvé dans Table1
   Table2.Rechercher(K2, Ma_Cle, Succes);
   Table2.Changer(K2, 0);
   -- on est sûr de trouver Table2(K2) = Ma_Cle et on libère la place
   V(Mutex_Entrepot);
   -- on compte une place libre de plus
end Supprimer;


procedure Modifier (D : in Document; Ma_Cle : in Cle) is
   K1 : Table1.Index; K2 : Table2.Index; Succes : boolean;
begin
   P(Mutex_Entrepot);
   Table1.Rechercher(K1, Ma_Cle, Succes);
   -- recherche K1 tel que Table1(K1) = Ma_Cle
   if Succes then
      -- la recherche est Succes
      Stock1(K1) := D;
      -- on écrit D dans stock1 et c'est fini
      V(Mutex_Entrepot);
      return;
      -- sortie de la procedure Modifier
   end if;
   Table2.Rechercher(K2, Ma_Cle, Succes);
   -- on est sûr de trouver Table2(K2) = Ma_Cle
   Stock2(K2) := D;
   -- on écrit D dans stock2 et c'est fini
   V(Mutex_Entrepot);
end Modifier;

begin
   -- initialisation du package Entrepot : initialisation des sémaphores
   E0(Mutex_Entrepot,1);
end Entrepot;
```

Question 1 (5 pts)

On suppose que ce paquetage doit être utilisé par des processus concurrents et, pour accroître la concurrence d'accès, on veut décomposer `Entrepot` en trois ressources critiques qui sont :

1) l'entier `Numero`, servant à associer une clé à chaque document

2) `Stock1`, sa table d'occupation `Table1` et leurs données de gestion,

3) `Stock2`, sa table d'occupation `Table2` et leurs données de gestion.

On veut s'arranger pour que chacune de ces ressources soit utilisée dans une section critique spécifique.

On veut aussi tenir compte du fait que l'entrepôt ne peut contenir qu'un nombre fini de documents : quand il y en a déjà 5050, on impose que tout processus qui demande le dépôt d'un nouveau document reste bloqué jusqu'à ce qu'une place soit libérée par un autre processus.

On vous demande d'améliorer la programmation en introduisant dans l'annexe 1 (qui vous est fournie ci-après) des sémaphores qui remplacent l'unique `Mutex_Entrepot` *(par exemple* Mutex_Numero, Mutex1, Mutex2, S_Stock*) pour prendre en compte cette concurrence accrue et cette limitation à 5050 places.*

Transaction

Le cahier des charges de l'entreprise stipule aussi que seules trois sortes de transactions sont envisagées, ce qui s'exprime par la spécification du paquetage suivant, avec les trois procédures correspondant chacune à une sorte de transaction :

```
package Transaction is
   type Liste_Cle is array (1..5) of Cle;
   procedure Suite_Consultations (Nb: in positive; S: in Liste_Cle);
   procedure Suite_avec_Modification_Suppression (Nb: in positive; S: in Liste_Cle);
   procedure Creation (D: in Document; Ma_Cle: out Cle);
end Transaction;
```

La programmation de ce paquetage `Transaction` a été faite elle aussi en sous-traitance en supposant que toutes les procédures (donc toutes les transactions) étaient exécutées en exclusion mutuelle entre elles. Le résultat fourni est :

```
with SRI_B; use SRI_B; with Entrepot; use Entrepot;
package body Transaction is
   Mutex_Transaction : semaphore;
procedure Suite_Consultations (Nb : in positive; S : in Liste_Cle) is
   Doc : Document;
begin
   P(Mutex_Transaction);
   for I in 1..Nb loop
      Consulter (Doc, S(I));
      -- consultation de Nb documents
      end loop;
   V(Mutex_Transaction);
end Suite_Consultations;

procedure Suite_avec_Modification_Suppression (Nb : in positive; S : in Liste_Cle)
is
begin
   P(Mutex_Transaction);
   for I in 1..Nb loop
      Consulter(Doc, S(I));
      - consultation de Nb documents
      Modifier(Doc, S(I));
      -- modification du document consulte
   end loop;
   for I in 2..Nb-1 loop
      Supprimer(Doc, S(I));
      -- suppression de document
   end loop;
   V(Mutex_Transaction);
end Suite_avec_Modification_Suppression;

procedure Creation (D : in Document; Ma_Cle : out Cle) is
begin
   P(Mutex_Transaction);
   Déposer (D, Ma_Cle);
   V(Mutex_Transaction);
end Creation;

begin
   -- initialisation du package Transaction : initialisation des sémaphores
   E0(Mutex_Transaction, 1);
end Transaction;
```

Le programme principal qui décrit l'utilisation du paquetage figure pour information en fin du texte dans la partie "compléments de programmation".

Question 2 (5 pts)

On suppose maintenant que ce paquetage doit être utilisé pour des transactions concurrentes selon les règles suivantes :

- soit les utilisateurs font des suites de consultations de documents dans l'entrepôt, en utilisant la procédure `Suite_Consultations.` Plusieurs transactions `Suite_Consultations` doivent pouvoir se faire en parallèle,
- soit l'administrateur fait une mise à jour entraînant des modifications ou des suppressions de documents. Cette transaction de l'administrateur qui utilise la procédure `Suite_avec_Modification_Suppression` doit s'exclure avec les transactions `Suite_Consultations`, et ne doit pas attendre indéfiniment la fin des transactions des utilisateurs,
- soit les concepteurs créent de nouveaux documents. Ces transactions qui utilisent la procédure `Creation` peuvent se faire en concurrence avec les transactions des utilisateurs ou avec la transaction de l'administrateur, car celles-ci ne peuvent utiliser que des clés qui ne sont connues qu'une fois le document créé. On limite cependant à 5 le nombre de dépots concurrents par des exécutions de la procédure `Creation`.

Autrement dit, les concepteurs ont toujours accès à l'entrepôt, mais 5 au plus en même temps ; les utilisateurs y accèdent en parallèle quand l'administrateur n'y accède pas ; dès que l'administrateur veut faire une transaction, les utilisateurs qui démarrent de nouvelles transactions doivent être bloqués et l'administrateur doit pouvoir faire sa transaction dès que toutes les transactions utilisateurs en cours sont terminées.

On vous demande de compléter le travail du sous-traitant en introduisant dans l'annexe 2 (qui vous est fournie ci-après) des données et des sémaphores qui remplacent l'unique `Mutex_Transaction` *(par exemple* `S_Fifo`, `Mutex_Consultation`, `S_Depot`, `S_ModSupp`*) pour prendre en compte cette concurrence.*

Va et vient entre Stock1 et Stock2

On rappelle que l'implémentation de cet entrepôt utilise deux tableaux de documents :

- Stock1, qui est de petite taille et d'accès rapide, et que l'on remplit en premier,

- Stock2, qui est de très grande taille (5000 documents) et d'accès plus lent, et que l'on remplit quand il n'y a plus de place dans Stock1.

Quand Stock1 est plein, on doit utiliser Stock2.

On veut étudier quelques politiques de placement et de va et vient des documents entre Stock1 et Stock2.

Pour un document i, il y aura plusieurs actions : le dépôt di (dans l'entrepôt), l'accès ai pour consulter ou modifier, la suppression si (dans l'entrepôt).

On dit que le document i est plus ancien que le document j si i<j (et plus récent si i>j). On suppose que le dépôt ou l'accès prend 1 unité de temps pour Stock1 et que le dépôt ou l'accès prend x unités de temps pour Stock2. Les suppressions prennent un temps nul. Tout transfert de Stock1 à Stock2 ou vice-versa prend x unités de temps.

Pour faire des tests, on se donne une trace d'utilisation de l'entrepôt .

trace: d1, d2, d3, d4, d5, a1, a3, d6, a5, a2, d7, a4, s3, a6, d8, a7, d9, s1, a9, a7, s6, a8, a9, a5, a4, a5, a9, a5

On veut tester plusieurs politiques avec cette trace pour le cas où Stock1 ne pourrait contenir que 4 documents (On donnera à x les valeurs 10 et 100 pour se faire deux idées) :

a) **le placement statique sans migration** (c'est ce qui est fait dans la question 1). Il vient :

trace	d1	d2	d3	d4	d5	a1	a3	d6	a5	a2	d7	a4	s3	a6	d8	a7	d9
Stock1	1	1 2	1 2 3	1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4	1 2 3 4	1 2 4	1 2 4	1 2 4 8	1 2 4 8	1 2 4 8
Stock2					5	5	5	5 6	5 6	5 6	5 6 7	5 6 7	5 6 7	5 6 7	5 6 7	5 6 7	5 6 7 9
Coût	1	1	1	1	x	1	1	x	x	1	x	1		x	1	x	x

trace (suite)	s1	a9	a7	s6	a8	a9	a5	a4	a5	a9	a5
Stock1	2 4 8	2 4 8	2 4 8	2 4 8	2 4 8	2 4 8	2 4 8	2 4 8	2 4 8	2 4 8	2 4 8
Stock2	5 6 7 9	5 6 7 9	5 6 7 9	5 7 9	5 7 9	5 7 9	5 7 9	5 7 9	5 7 9	5 7 9	5 7 9
Coût		x	x		1	x	x	1	x	x	x

coût total = 11 + 14x (soit 151 avec x = 10 et 1411 avec x = 100)

b) **le placement dynamique en plaçant les plus récents documents dans Stock1**. Tout dépôt di place le document i dans Stock1 en chassant, si nécessaire, vers Stock2 le document j le plus ancien de Stock1. Les accès aj et les suppressions sj se font dans Stock1 ou dans Stock2, là où est placé le document j. En cas de suppression dans Stock1, on laisse la place vide, on ne remonte pas de document depuis Stock2.

trace	d1	d2	d3	d4	d5	a1	a3	d6	a5	a2	d7	a4	s3	a6	d8	a7	d9	s1	a9
Stock1	1	1 2	1 2 3	1 2 3 4	2 3 4 5	2 3 4 5	2 3 4 5	3 4 5 6	3 4 5 6	3 4 5 6	4 5 6 7	4 5 6 7	4 5 6 7						
Stock2					1	1	1	1 2	1 2	1 2	1 2 3	1 2 3	1 2						
S1->S2					1			2			3								
Coût	1	1	1	1	x+1	x	1	x+1	1	x	x+1	1							

c) **le remplacement dynamique dans Stock1.** Tout dépôt di se fait dans Stock1 et, si Stock1 st plein, le document i remplace le plus ancien document j, tout accès ai fait remonter le document i dans Stock1 où il remplace le plus ancien document j de Stock1. Le document j va dans Stock2. En cas de suppression dans Stock1, on laisse la place vide, on ne remonte pas de document depuis Stock2.

trace	d1	d2	d3	d4	d5	a1	a3	d6	a5	a2	d7	a4	s3	a6	d8	a7	d9	s1	a9
Stock1	1	1 2	1 2 3	1 2 3 4	2 3 4 5	1 3 4 5	1 3 4 5	3 4 5 6	3 4 5 6	2 4 5 6	4 5 6 7	4 5 6 7	4 5 6 7						
Stock2					1	2	2	1 2	1 2	1 3	1 2 3	1 2 3	1 2						
S2->S1						1				2									
S1->S2					1	2		1		3	2								
Coût	1	1	1	1	1+x	2x+1	1	1+x	1	2x+1	1+x	1							

Question 3 (3 pts)

On vous a préparé l'annexe 3 que vous compléterez pour comparer les 3 politiques. (On donnera à x les valeurs 10 et 100 pour se faire deux idées). Quelle conclusion en déduisez vous ? Expliquer pourquoi ces résultats sont mauvais.

Question 4 (4 pts)

En vous inspirant des politiques choisies pour la pagination à la demande, proposer une autre politique de remplacement (annexe 4) qui devrait être meilleure. Expliquer pourquoi et tester sur la trace. Est ce que cela a donné le résultat souhaité ? Expliquez pourquoi cela n'a pas marché. Qu'est ce qu'il faudrait faire pour avoir de meilleurs résultats ?

On agrandit Stock1 pour qu'il puisse contenir 5 documents. On ajoute 25% de place en plus et on constate une amélioration pour la politique de gestion a) Placement statique sans migration

trace	d1	d2	d3	d4	d5	a1	a3	d6	a5	a2	d7	a4	s3	a6	d8	a7	d9
Stock1	1	1 2	1 2 3	1 2 3 4	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 3 4 5	1 2 4 5	1 2 4 5	1 2 4 5 8	1 2 4 5 8	1 2 4 5 8
Stock2								6	6	6	6 7	6 7	6 7	6 7	6 7	6 7	6 7 9
S2->S1																	
S1->S2																	
Coût	1	1	1	1	1	1	1	x	1	1	x	1		x	1	x	x

trace	s1	a9	a7	s6	a8	a9	a5	a4	a5	a9	a5
Stock1	2 4 5 8	2 4 5 8	2 4 5 8	2 4 5 8	2 4 5 8	2 4 5 8	2 4 5 8	2 4 5 8	2 4 5 8	2 4 5 8	2 4 5 8
Stock2	6 7 9	6 7 9	6 7 9	7 9	7 9	7 9	7 9	7 9	7 9	7 9	7 9
S2->S1											
S1->S2											
Coût		x	x		1	x	1	1	1	x	1

coût total = 16 + 9x (soit 106 et 916)

Le coût d'accès au magasin qui était de 11 + 14x passe à 16 + 9x.

Le coût d'accès à Stock2 baisse de (14x - 9x)/14x = 36%. Le coût d'accès au magasin baisse de

(151 - 106)/151 = 30% pour x = 10

(14 11- 916)/1411 = 35% pour x = 100

Question 5 (3 pts)

Remplir le tableau de l'annexe 4 pour la politique que vous avez donné à la question 4 (gestion d). Quel gain obtient-on ? Peut-on extrapoler linéairement ces résultats aux cas où on ajouterait une sixième place à Stock1 puis une septième place.

Vous expliquerez votre réponse à la lueur des résultats connus dans les systèmes informatiques.

Démon de minuit (question optionnelle)

Quand `Stock1` est plein, on doit utiliser `Stock2`. On a étudié des politiques de va et vient entre `Stock1` et `Stock2`. En attendant le résultat de cette étude, on veut conserver quelques places libres dans `Stock1`. Pour cela, on a fait programmer, dans le paquetage `Entrepot`, un `Demon` qui recherche périodiquement le plus ancien document de `Stock1` (par recherche de clé dans `Table1`) pour le transférer dans `Stock2` s'il y a de la place pour le faire. Ici à nouveau, on a fait programmer le démon avec un seul sémaphore `Mutex_Entrepot`. Le résultat est ceci :

DÉMON DE L'ENTREPOT

```
-- le démon déplace le plus ancien document de Table1 vers Table2 si on peut, sinon on ne
fait rien
task Demon;
task body Demon is
   Minuit : duration := 0.1; -- simule le réveil à minuit
   Plus_Petite_Cle : Positive := 1; -- il n'y a pas de clé plus petite dans Table1
   K1 : Table1.Index; K2 : Table2.Index; Succes;
begin
   loop
      delay Minuit; -- lancer le Démon de minuit
      P(Mutex_Entrepot);
      if Table1.Place < N1/2 then
         -- il n'y a pas trop de places vides dans Stock1
         while not Succes loop
            -- La clé est-elle encore présente?
            Table1.Rechercher(K1, Plus_Petite_Cle , Succes);
            Plus_Petite_Cle := Plus_Petite_Cle + 1;
            -- sinon on recherche la clé suivante
            end loop;
         Table2.Rechercher(K2, 0, Succes); -- on recherche une place libre, c'est K2
         if Succes then
         -- Table2 n'est pas pleine
            Table2.Changer(K2, Mon_Numero);
            Stock2(K2) := Stock1(K1);-- on ne déplace que si K2 existe
            Table1.Changer(K1, 0);
         else -- Table2 est pleine
            -- on doit corriger pour la fois suivante
            Plus_Petite_Cle := Plus_Petite_Cle - 1;
            -- s'il y a trop de places vides, on ne déplace pas de document cette fois
         end if;
      end if;
      V(Mutex_Entrepot);
   end loop;
end Demon;
```

Question 6 (3 pts)

On vous demande de compléter ce travail en introduisant dans l'annexe 5 (qui vous est fournie ci-après) des données et des sémaphores qui remplacent l'unique `Mutex_Entrepot` *conformément à ce qui a été fait dans la question 1 pour utiliser plusieurs ressources critiques au lieu d'une. Mais cette fois, on doit emboiter l'utilisation de deux ressources critiques.*

On veillera attentivement à éviter le fonctionnement erroné suivant (avec les deux ressources critiques Table1 + Stock1 et Table2 + Stock2) :

1) à l'instant t, un nouvel appel de Deposer a été autorisé à se poursuivre car il reste encore une place libre (qui, à t, est située dans Table2). Deposer ne trouve pas de place dans Table1, il libère Table1 et s'apprête à chercher une place dans Table2.

2) à t+1 le démon est activé et, comme Table1 lui est accessible, le démon trouve la plus petite clé de Table1 et passe sur Table2. Puis il recopie un document de Stock1 dans Stock2 en utilisant la dernière place libre de Table2. La place libre est maintenant dans Table1. Table2 est libérée par le démon.

3) à t+2, Déposer a accès à Table2 qu'il trouve pleine ; on a "Succes = False" et "K2 = 5000". Comme cela n'est pas prévu, Déposer écrase le document dont la clé est en Table2(K2) en exécutant Stock2(K2) := D.

(Et on passera des mois à rechercher cette erreur qui pourra rester cachée longtemps!!)

Si vous ne trouvez pas la solution, relisez plutôt attentivement vos réponses aux autres questions pour y corriger les erreurs d'inattention. Les cinq premières questions permettent d'avoir 20.

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

```
procedure Exam_99_06 is -- PROGRAMME PRICIPAL
   Liste_Commune : Liste_Cle; Un_Moment : duration := 0.05;
   task type Concepteur; task type Utilisateur;
   task body Concepteur is
      D : Document; Ma_Cle : Cle;
   begin loop Creation (D, Ma_Cle); delay (10 * Un_Moment); end loop;
   end Concepteur;
   task body Utilisateur is
      Nb_U : Integer := 1;
   begin loop
      Suite_Consultations(Nb_U, Liste_Commune); Nb_U := Nb_U mod 5 + 1;
      delay Un_Moment;
   end loop;
   end Utilisateur;
   Le_Concepteur: array (1..10) of Concepteur;
   L_Utilisateur: array (1..100) of Utilisateur;
   task Administrateur;
   task body Administrateur is
      Nb_A : Integer := 1;
   begin loop
      Suite_avec_Modification_Suppression (Nb_A, Liste_Commune); Nb_A := Nb_A mod 5 + 1 ;
      delay (100 * Un_Moment);
   end loop;
   end Administrateur;
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
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 => Element'First); --initialise Tab avec Element'First
   procedure Rechercher(I : out Index; B : in Element; Reussi : out boolean) is
      J : Index := 1; -- on sait queTab 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 Place return Natural is -- indique le nombre de places libres, repérées par Element'First
      J : Index := 1; K : Integer;
   begin
      while J < Taille loop
         if Tab(J) = Element'First then K := K + 1; end if; J := J + 1;
      end loop;                                          -- il reste à examiner le contenu du dernier indice
      if Tab(J) = Element'First then K := K + 1; end if;
      return K;
   end Place;
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.

```
--------------------------  SRI_B importe les sémaphores et les opérations P, V et E0
with SRI_B; use SRI_B; with Table_Generique; use Table_Generique;
package body Entrepot is  ------------------------------------------------- ENTREPOT
   N1 : constant Positive := 50; N2 : constant Positive := 5000;
   N : constant := N1 + N2;                    -- total des places dans l'entrepôt
   subtype Index1 is Positive range 1..N1; subtype Index2 is Positive range 1..N2;
----------------  données communes partagées par les procédures du paquetage
   Stock1 : array (Index1) of Document; Stock2 : array (Index2) of Document;
   package Table1 is new Table_Generique (taille => N1, Element => Natural);
   package Table2 is new Table_Generique (taille => N2, Element => Natural);
   Numero : Natural := 0;  -- sert ànumératoter les clés de façon unique jusqu' à Integer'Last
                                                  : Semaphore;
-------------------------------------------------------------------------  ENTREPOT
   procedure Deposer (D : in Document; Ma_Cle : out Cle) is
      K1 : Table1.Index; K2 : Table2.Index; Succes : boolean;
   begin
                    -- acquisition d'une cle unique pour ce nouveau document entreposé

      Numero := Numero + 1; Ma_Cle := Numero;

   -- Ya-t-il assez de place dans Table1 ou Table2 et Stock1 ou Stock2 ?


      Table1.Rechercher(K1, 0, Succes);  -- recherche une place K1 telle que  Table1(K1) = 0
      if Succes then                                  -- la recherche est  réussie
         Table1.Changer(K1, Ma_Cle);       -- on note l'occupation dans Table1 avec Ma_Cle
         Stock1(K1) := D;                    -- on écrit dans stock1 et c'est fini

         return;                       -- sortie de la procedure Deposer
      end if;



                                    -- on n'a pas trouvé dans Table1, on recherche dans Table2
      Table2.Rechercher(K2, 0, Succes);  -- recherche une place K2 telle que Table2(K2) = 0 ;
      Table2.Changer(K2, Ma_Cle );          -- on est sûr de trouver une place dans Table2
      Stock2(K2) := D;                      --on écrit dans Stock2 et c'est fini

   end Deposer ;
-------------------------------------------------------------------------------  ENTREPOT
   procedure Consulter (D : out Document; Ma_Cle : in Cle) is
      K1 : Table1.Index; K2 : Table2.Index; Succes : boolean;
   begin

      Table1.Rechercher(K1, Ma_Cle, Succes); -- recherche K1 tel que Table1(K1) = Ma_Cle
      if Succes then                          -- on a trouvé, la recherche est Succes
         D := Stock1(K1);                     --on lit dans stock1 et c'est fini

         return;                       -- sortie de la procedure Consulter
      end if;

                              -- on n'a pas trouvé dans Table1, on cherche dans Table2
      Table2.Rechercher(K2, Ma_Cle, Succes); -- on est sûr de trouver Table2(K2) = Ma_Cle
      D := Stock2(K2);                          --on lit dans stock2 et c'est fini

   end Consulter;
```

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

```
---------------------------------------------------------------------------------- ENTREPOT
   procedure Supprimer (Ma_Cle : in Cle) is
      K1 : Table1.Index; K2 : Table2.Index; Succes : boolean;
   begin

      Table1.Rechercher(K1, Ma_Cle, Succes);  -- recherche K1 tel que Table1(K1) = Ma_Cle
      if Succes then                          -- la recherche est un Succes
         Table1.Changer(K1, 0);               -- on libère la place en inscrivant 0

         return;                              -- sortie de la procedure Supprimer
      end if;


      Table2.Rechercher(K2, Ma_Cle, Succes);        --  on n'a pas trouvé dans Table1
               -- on est sûr de trouver Table2(K2) = Ma_Cle et on libère la place
      Table2.Changer(K2, 0);

                                       -- on compte une place libre de plus
   end Supprimer;
---------------------------------------------------------------------------------- ENTREPOT
   procedure Modifier (D : in Document; Ma_Cle : in Cle) is
      K1 : Table1.Index; K2 : Table2.Index; Succes : boolean;
   begin

      Table1.Rechercher(K1, Ma_Cle, Succes);-- recherche K1 tel que Table1(K1) = Ma_Cle
      if Succes then                          -- la recherche est Succes
         Stock1(K1) := D;                     --on écrit D dans stock1 et c'est fini

         return;                              -- sortie de la procedure Modifier
      end if;


      Table2.Rechercher(K2, Ma_Cle, Succes); -- on est sûr de trouver Table2(K2) = Ma_Cle
      Stock2(K2) := D;                       --on écrit D dans stock2 et c'est fini

   end Modifier;
-------------------
begin

                                          --initialisation des sémaphores
end Entrepot;  ----------------------------------------------------------------ENTREPOT

-------------------------------------------------------------------------------------------------
```

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

```
with SRI_B; use SRI_B; with Entrepot; use Entrepot;
package body Transaction is ------------------------------------------
   TRANSACTION
                                                        : semaphore;
------------------
   procedure Suite_Consultations (Nb : in positive; S : in Liste_Cle) is
      Doc : Document;
   begin




      for I in 1..Nb loop
         Consulter (Doc, S(I));              -- consultation de Nb documents
      end loop;



   end Suite_Consultations;
-------------------------------------------------------------- TRANSACTION
   procedure Suite_avec_Modification_Suppression
                                        (Nb : in positive; S : in Liste_Cle) is
   begin




      for I in 1..Nb loop
         Consulter(Doc, S(I));           - - consultation de Nb documents
         Modifier(Doc, S(I));              -- modification du document consulte
      end loop;
      for I in 2..Nb-1 loop
         Supprimer(Doc, S(I));               -- suppression de document
      end loop;



   end Suite_avec_Modification_Suppression;
-------------------------------------------------------------- TRANSACTION
   procedure Creation (D : in Document; Ma_Cle : out Cle) is
   begin


      Déposer (D, Ma_Cle);


   end Creation;
-------------------------------------------------------------- TRANSACTION
begin                                  -- initialisation des sémaphores etc...

end Transaction;---------------------------------------------- TRANSACTION
```

ANNEXE 5 *Pour l'anonymat* ***, Mettez ici votre numéro de copie*** *:*
Ne mettez ni nom, ni numéro de carte CNAM.

```
---------------------------------------------------- DÉMON DE L'ENTREPOT
-- le démon déplace le plus ancien document de Table1 vers Table2 si on peut,
-- sinon on ne fait rien
   task Demon;
   task body Demon is
      Minuit : duration := 0.1;                    --simule le réveil à minuit
      Plus_Petite_Cle : Positive := 1;             -- il n'y a pas de clé plus petite dans
Table1
      K1 : Table1.Index; K2 : Table2.Index; Succes;
   begin
   loop
      delay Minuit;                                -- lancer le Démon de minuit


      if Table1.Place < N1/2 then                  -- il n'y a pas trop de places vides dans Stock1
         while not Succes loop
           Table1.Rechercher(K1, Plus_Petite_Cle , Succes); -- La clé est-elle encore
présente?
           Plus_Petite_Cle := Plus_Petite_Cle + 1;      -- sinon on recherche la clé suivante
           end loop;                                    -- la recherche a-t-elle abouti?


         Table2.Rechercher(K2, 0, Succes);    -- on recherche une place libre, c'est K2
         if Succes then                                -- Table2 n'est pas pleine
            Table2.Changer(K2, Mon_Numero);
            Stock2(K2) := Stock1(K1);              -- on ne déplace que si K2 existe
            Table1.Changer(K1, 0);
         else                                                    -- Table2 est pleine
            Plus_Petite_Cle := Plus_Petite_Cle - 1; -- on doit corriger pour la fois suivante
         end if;

      end if;-- s'il y a trop de places vides, on ne déplace pas de document  cette fois


   end loop;
   end Demon; ----------------------------------------------------DÉMON DE L'ENTREPOT
```