SYSTÈMES INFORMATIQUES ET APPLICATIONS CONCURRENTES

COURS B4 : HTO (NFP137) et ICPJ (NFP137J)

CORRIGÉ DE L'EXAMEN DU 24 JUIN 2006

Barème indicatif

Question	R1	R2	R3	R4	S1	S2	S3	S4	S5	S6
Notation	2	1	2	1	3	4	1	2	2	2

SOLUTIONS

QUESTION R1 : Parmi les 12 requêtes de 3 ressources, 10 peuvent être servies. Il reste alors 6 ressources, ce qui permet de servir toutes les requêtes possibles d'un processus qui ferait une commande de mise à jour, puis, cette mise à jour terminée et les 9 ressources récupérées, de servir toutes les autres commandes, éventuellement l'une après l'autre (dans le pire cas, ce sont toutes des mises à jour) une fois celle-ci terminée.

QUESTION R2 : On applique le calcul de la politique de précaution avec $X > \sum(Ci - 1)$. $Ci = C = 9$. Le nombre minimal de ressources qui permette de garantir l'absence d'interblocage est alors $12*8 + 1 = 97$

QUESTION R3 : 11 processus peuvent recevoir leur première requête de 3 ressources, car il reste 3 ressources pour servir une commande de lecture (il y en a au moins 5 qui peuvent être servies, sur ces 11) et celle-ci terminée, on aurait les 6 ressources pour servir une requête de mise à jour ou un autre requête de lecture, donc toutes les autres requêtes.

QUESTION R4 : On applique le calcul de la politique de précaution avec $X > \sum(Ci - 1)$. On a deux annonces possibles $C1 = 6$ et $C2 = 9$. Le nombre minimal de ressources qui permette de garantir l'absence d'interblocage est alors $6*5 + 6*8 + 1 = 79$

QUESTION S1 : -- solution avec le paradigme lecteurs-rédacteurs ; possible aussi avec sémaphores privés

```
package body ControleBase is
    S_Cohorte : Semaphore;-- pour garantir qu'il n'y a pas plus de 10 consultations simultanément
    Mutex_A, Mutex_L : Semaphore; -- déclaration des sémaphores de contrôle des accès
    NL : Natural := 0; -- déclaration et initialisation des variables rémanentes utilisées pour ce contrôle
    procedure AccesConsultation is
    begin
        P(S_Cohorte) ;       -- régulation et blocage si plus de 10 accès
        -- contrôle de l'accès en consultation et blocage s'il y a en cours un accès en mise à jour
        P(Mutex_L); NL := NL + 1; if NL = 1 then P(Mutex_A); end if; V(Mutex_L);
    end AccesConsultation ;
    procedure SortieConsultation  is
    begin
        -- sortie de l'utilisation en consultation et gestion des conséquences de cette sortie
        P(Mutex_L); NL := NL - 1; if NL = 0 then V(Mutex_A); end if; V(Mutex_L);
        V(S_Cohorte) ;       -- régulation des 10 accès simultanés et réveil éventuel d'un processus en attente
    end SortieConsultation ;
    procedure AccesMiseAJour  is
    begin
        P(Mutex_A); -- contrôle de l'accès en mise à jour, respectant la cohérence des consultations et des mises à jour
    end AccesMiseAJour;
    procedure SortieMiseAJour
    begin
        V(Mutex_A); -- sortie de l'utilisation en mise à jour et gestion des conséquences de cette sortie
    end SortieMiseAJour ;
begin
    E0(S_Cohorte , 10); E0(Mutex_A, 1); E0(Mutex_L, 1); -- initialisation des sémaphores
end ControleBase;
```

QUESTION S2

```
package body ControleRessource is
    Stock : Integer := StockInitial; -- variable rémanente partagée, donnée de contrôle
```

```
    EnAttente : Boolean := False; -- variable rémanente partagée, donnée de contrôle
    S_Solo, S_Patience : Semaphore; -- pour garantir l'unicité du client servi, et pour le faire attendre si nécessaire
    Mutex : Semaphore; -- déclaration des autres sémaphores nécessaires
    procedure Demander(X : in Integer) is
        OK : boolean := False; -- variable non partagée, créée à chaque appel de cette procédure, détruite au retour de l'appel
    begin
        P(S_Solo); -- Assurer avec Solo que l'on ne sert bien chaque client l'un après l'autre
        P(Mutex); --Assurer la cohérence des données rémanentes de contrôle et le non blocage en section critique
        Stock := Stock – X; -- le résultat est négatif s'il n'y a pas assez de ressources en Stock
        if Stock < 0 then
            EnAttente:= True; OK := False;
        else
            EnAttente:= False; OK := True;
        end if;
        V(Mutex);
        if OK then
            V(S_Solo);  -- Autoriser un autre client à examiner sa demande
        else
            P(S_Patience); -- Mettre le client demandeur en attente de retour de ressources
        end if;
    end Demander;
    procedure Restituer(Y : in Integer) is
    begin
        P(Mutex); -- assurer la cohérence des données rémanentes de contrôle
        Stock := Stock + Y;
        if EnAttente and Stock >= 0 then
            EnAttente := False;
            V(S_Patience) ; -- Réveiller le processus en attente de retour de ressource
            V(S_Solo) ; -- Autoriser un autre client à examiner sa demande dans Demander
        end if;
        V(Mutex) ;
    end Restituer;
begin
    E0(Mutex, 1); E0(S_Solo, 1); E0(S_Patience, 0); -- initialisation des sémaphores et des variables de contrôle
end ControleRessource;
```

QUESTION S3 : Le sémaphore S_Patience, est réutilisé par chaque processus qui demande en solo. Il faut s'assurer que lorsque ce sémaphore S_Patience, est réutilisé par un autre processus, il est bien revenu à son état initial avec son compte S_Patience.E à 0 avant cette réutilisation. Sinon si son compte S_Patience.E est encore -1 au moment de l'opération P(S_Patience), cela veut dire que le processus en attente de retour de ressource a été servi, mais pas encore réveillé. Et il y aura alors deux processus bloqués par S_Patience, l'ancien qui a été servi et un nouveau qui ne peut être servi. Comme rien ne garantit que ce sera le premier bloqué qui sera libéré le premier (la gestion de la file d'attente S_Patience.F n'est pas obligatoirement FIFO. Elle est souvent gérée avec des priorités), quand le prochain v(S_Patience) sera exécuté, ce ne sera pas obligatoirement l'ancien qui sera libéré, le nouveau pourrait l'être à sa place. Dans le programme, il faut veiller à toujours laisser V(S_Patience) avant V(S_Solo).
Au niveau de la programmation, cela autorise à faire l'optimisation suivante

```
    procedure Restituer(Y : in Integer) is
    begin
        P(Mutex); -- assurer la cohérence des données rémanentes de contrôle
        Stock := Stock + Y;
        if EnAttente and Stock >= 0 then
            EnAttente := False; V(Mutex);
            V(S_Patience) ; -- Réveiller le processus en attente de retour de ressource donc S_Patience.E = 0
            V(S_Solo) ; -- Autoriser un autre client à examiner sa demande dans Demander
        else
            V(Mutex);
        end if;
    end Restituer;
```

mais interdit l'ordre de programmation suivant

```
    procedure Restituer(Y : in Integer) is
    begin
```

```
        P(Mutex); -- assurer la cohérence des données rémanentes de contrôle
        Stock := Stock + Y;
        if EnAttente and Stock >= 0 then
            EnAttente := False; V(Mutex);
            V(S_Solo) ; -- Autoriser un autre client à examiner sa demande dans Demander
-- s'il y a réallocation du processeur un processus appelant Demander peut franchir P(S_Solo) et P(Mutex) et se bloquer sur
-- P(S_Patience). On a alors deux processus bloqués sur ce sémaphore, d'où l'erreur possible si pas de gestion FIFO
            V(S_Patience) ; -- Réveiller le processus en attente de retour de ressource
        else
            V(Mutex);
        end if;
    end Restituer;
```

QUESTION S4 : Dans cette architecture 2.1, les 15 serveurs peuvent demander toutes leurs ressources avant tout accès à la base. On applique le calcul de la politique de précaution avec $X > \sum(Ci - 1)$. On a deux annonces possibles C1= 6 et C2 = 8. Le nombre minimal de ressources qui permette de garantir l'absence d'interblocage est alors StockInitial = 12*5 + 3*7 + 1 = 82 ;
La concurrence maximale est obtenue pour StockInitial = 12*6 + 3*8 = 96

QUESTION S5 : Dans cette architecture 2.2, les serveurs ne peuvent demander de ressources que lorsqu'ils ont accès à la Base. Donc soit il y a 10 consultations et le minimum de ressources est alors 10*5 + 1. Ou bien il y a une mise à jour qui peut demander 8 ressources. Le nombre minimal de ressources nécessaire doit permettre de garantir l'absence d'interblocage, que la base soit en consultation ou qu'elle soit en mise à jour. Cette valeur minimale est alors StockInitial = max((10*5 + 1), 8) = 51.
Pour obtenir la concurrence maximale lors d'une consultation, il faut 10*6 = 60 ressources initialement.

QUESTION S6 : Dans cette architecture 2.3, les serveurs de consultation ne peuvent demander de ressources que lorsqu'ils ont accès à la Base. Donc soit il y a 10 consultations et le minimum de ressources est alors 10*5 + 1. Mais les serveurs de mise à jour peuvent chacun demander leurs 8 ressources avant de faire l'accès à la base (un seul y a accès, ils feront ces accès l'un après l'autre). Au total les serveurs de mise à jour peuvent demander 3*8 = 24 ressources.
Comme on peut avoir à servir ces 10 consultations alors que les 3 mises à jour ont déjà obtenu toutes leurs ressources, il faut une valeur minimale qui met ces consultations à l'abri d'un interblocage. D'où
StockInitial = 3*8 + 10*5 + 1 = 75
Concurrence dépendant de StockInitial. Le nombre de consultations concurrentes maximale sera toujours 10 si StockInitial >= 3*8 + 10*6 = 84
Interblocage croisé si les 3 serveurs de mise à jour ont consommé 3*8 = 24 ressources et si ce sont les serveurs de consultation qui ont l'accès à la base et si aucun de ces serveurs ne peut avoir les 6 ressources qu'il peut demander. On a vu que cela se produirait si StockInitial <75.
Donc si StockInitial < 75, il faut mettre un algorithme de prévention d'interblocage sur l'allocation des ressources (algorithme du banquier) et celui-ci empêchera cette situation d'allocation de ressources, mais n'empêchera pas l'interblocage croisé (il faudrait une prévention avec deux classes de ressource : les ressources et la base documentaire). Ou alors il faut garantir que si les trois serveurs de mise à jour sont servis, on peut servir au moins un serveur de consultation. d'où StockInitial >= (3*8 + 6) = 54.