CHAPITRE 6

INTERBLOCAGE

Plan

Manuel 6

Interblocage

1 Conditions d'apparition du phénomène

La concurrence entre processus et l'allocation dynamique de ressources peuvent introduire un interblocage. Ce phénomène résulte de la conjonction de quatre circonstances.
1) Chaque ressource est utilisable exclusivement par un processus, et un seul, car son partage donnerait des résultats incohérents.
2) Chaque processus ne peut progresser que s'il obtient les ressources qu'il requiert dynamiquement par des demandes successives ; il attend donc les ressources de sa dernière requête avant de poursuivre son déroulement.

3) Les ressources déjà allouées restent nécessaires aux processus qui les ont reçue et elles ne peuvent être réquisitionnées.
4) Des processus sont en attente de ressources allouées à d'autres processus et il s'est formé une attente circulaire entre les processus.

Lors d'un interblocage, deux processus, ou davantage, sont en attente infinie, car ils ne peuvent sortir de cette attente, ni en partageant, ni en rendant, ni en réquisitionnant des ressources.

Si les processus qui attendent des ressources sont bloqués par l'allocateur, on est en interblocage passif ("deadlock"). S'ils attendent en exécutant cycliquement un test de disponibilité des ressources demandées, on est en interblocage actif ou étreinte fatale ("livelock").

Le cas d'interblocage le plus simple apparaît quand deux processus P1 et P2 partagent deux ressources R1 et R2 d'accès exclusif (fichiers en écriture, données système, catalogues) : P1 a obtenu R1 et demande R2 tandis que P2, qui a obtenu R2, demande R1. Un autre cas est celui de la demande dynamique de ressources banalisées de la même classe, chaque processus demandant moins du total disponible, mais la somme des demandes dépassant ce total.

2. Approches du problème

2.1. Détection guérison.

Des processus en attente ne sont pas obligatoirement en interblocage. Pour identifier celui-ci, il faut encore vérifier que l'attente est infinie. On doit enregistrer les demandes et les allocations et utiliser cette information pour détecter une attente circulaire. Cela revient à rechercher un cycle dans un graphe de n noeuds, recherche dont la complexité est en $O(n^2)$. Une fois ce cycle détecté, il faut le rompre en sacrifiant un processus en réquisitionnant ses ressources. Ceci n'est viable que si on sait ramener les ressources récupérées dans un état cohérent, ce qui peut nécessiter d'avoir posé préventivement des points de reprise . Le choix de la victime n'est jamais évident. Le plus logique est de sacrifier le processus le moins important pour l'application.

2.2. Prévention statique.

Pour ne pas avoir à détruire de processus, on essaie de mener une politique de prévention qui peut entraîner une sous utilisation des ressources. Dans la politique d'allocation globale, tout processus doit faire en une seule fois la demande de toutes ses ressources ; il les recevra en bloc même s'il ne s'en sert pas tout le temps. Les ressources peuvent être mal utilisées par le processus et le stock de ressources inutilisées peut être important si la demande non satisfaite est grande. Quand il y a plusieurs classes de ressources, l'allocation par classes ordonnées fractionne cette politique globale. Les classes sont ordonnées a priori selon un classement fixe, les ressources sont allouées selon cet ordre fixe, les ressources d'une classe sont servies en bloc. Une fois une classe servie, on ne peut revenir demander de ressource dans une classe antérieure. Cet ordre fixe d'allocation empêche la formation de cycle. Si les classes de ressources sont utilisées pendant des durées distinctes et si on peut les ordonner selon ce critère, alors l'utilisation des ressources peut être bonne.

Ces politiques sont déterminées dès la programmation et sont donc statiques.

2.3. Prévention dynamique (ou évitement) par attente et retard de service

Si on veut une politique de prévention qui soit dynamique et qui n'entraîne pas la destruction de processus, on doit estimer le comportement de ceux-ci par l'annonce du maximum de ressources que chacun peut demander. Grâce à cette annonce contractuelle, l'allocateur peut vérifier, avant de servir une nouvelle requête, s'il garde ou non la possibilité de répondre à toutes les demandes futures, même maximales, en les servant si nécessaire de façon séquentielle. S'il ne conserve pas cette possibilité, il y a risque d'interblocage et l'allocateur retarde la requête en bloquant le processus demandeur. L'allocateur pilote le système en l'obligeant à rester dans des états, appelés états fiables, à partir desquels le système ne peut jamais évoluer vers un interblocage. Cette contrainte peut amener un refus de service, même si l'allocateur a les ressources pour satisfaire la demande instantanée. Ce retard permet de ne pas

avoir à détruire de processus. Les ressources peuvent être sous utilisées si les annonces sont inégales. Par ailleurs, on n'est pas toujours en état de connaître toutes les ressources qu'un processus va utiliser et estimer un majorant conduirait à recenser toutes les ressources, ce qui peut s'avérer irréaliste dans une base de données par exemple.

2.4. Prévention destructive par estampilles

Dans les systèmes transactionnels, la pose de verrou d'accès à chaque ressource de la base de données peut entraîner un interblocage. Une méthode de prévention dynamique consiste à dater les transactions en leur associant une estampille et à utiliser l'ordre total ainsi obtenu pour régler le conflit d'accès aux ressources. Par exemple dans la méthode dite "Wait Die", la transaction qui bute sur un verrou n'est autorisée à attendre que si son estampille est inférieure à celle de la transaction qui a verrouillé la ressource ; sinon elle est abandonnée. Comme la transaction perdante doit être détruite, cette méthode ne peut fonctionner que si des points de reprise sont posés au démarrage de chaque transaction. Une transaction abandonnée redémarre avec son ancienne estampille, ce qui prévient la famine. Cette méthode s'applique quand l'identité des ressources accédées n'est pas connue au début de la transaction et qu'elle dépend de la navigation dans la base de données. On dit que cette méthode est pessimiste car la transaction perdante est abandonnée, même si elle n'utilise aucune ressource. Cette méthode s'applique aussi aux transactions réparties car on sait attribuer à chaque transaction une estampille unique dans l'application répartie.

2.5. Graphe de l'allocation de ressources

L'allocation de ressources peut être représentée par un digraphe où les processus et les ressources correspondent chacun à un type de noeud, où un processus qui détient une ressource est représenté par un arc de la ressource vers le processus, tandis que la demande d'un processus est figurée par un arc du processus vers la ressource. Dans cette représentation, un interblocage apparaît s'il y a un cycle dans le graphe.

3. Formalisation de l'interblocage

3.1. Représentation des processus et des ressources

Pour construire un algorithme efficace de prévention par retard de service, il faut formaliser le problème pour mieux le résoudre.

Le système informatique comprend n processus et m classes de ressources. Le nombre de ressources banalisées dans chaque classe constitue l'état initial du système. Cet état initial X, n et m sont fixes. Une ressource n'est utilisée que par un processus à la fois et elle est restituée au bout d'un temps fini. L'évolution du système est représentée par une matrice A(t) des ressources déjà allouées à la date t, et une matrice D(t) du cumul des ressources demandées à t. A(t) et D(t) se lisent aussi comme une suite de n vecteurs à m dimensions, Ai(t) et Di(t), qui représentent l'allocation et la demande de chaque processus i du système. Un vecteur R(t), appelé résidu, dénote les ressources non allouées, disponibles pour la politique de prévention. On a $R(t) = X - \sum Ai(t)$.

Le contrat, c'est à dire l'annonce de la demande maximale de ressources, est représentée par une matrice C, suite de n vecteurs Ci. On appelle distance, et on la note $Dist_i(t) = Ci - Ai(t)$, le nombre de ressources qu'un processus peut encore demander contractuellement.

Comme on va être amené à comparer des vecteurs ou de matrices, il faut définir rigoureusement les notations utilisées.

a) pour U, V vecteurs à m éléments : $U \leq V \Leftrightarrow ui \leq vi \quad i \in [1..m]$

$U < V \Leftrightarrow (U \leq V)$ et (i tel que $ui < vi$)

b) pour M, N matrices à $m * n$ éléments : $M \leq N \Leftrightarrow Mj \leq Nj \quad j \in [1..n]$

$M \leq N \Leftrightarrow Mj \leq Nj$ et (j tel que $Mj < Nj$)

3.2. États du système

Cette formalisation permet de classer les états parcourus pendant l'évolution du système, états réalisables qui correspondant à une réalité physique, états sains où le système n'est pas en interblocage, et états fiables d'où le système ne peut évoluer vers un interblocage.

3.3. État réalisable

La demande individuelle d'un processus et la somme des allocations, ne doivent pas dépasser les ressources initiales du système. Par ailleurs, on suppose que chaque processus reçoit au plus sa demande.

4. État sain

4.1. Recherche d'une suite saine complète

Pour montrer que le système n'est pas en interblocage à t ou n'est pas sur la voie de l'être, il faut montrer qu'avec les allocations déjà faites, l'allocateur, par une exécution séquentielle fictive, saurait satisfaire l'une après l'autre toutes les demandes actuelles non satisfaites. Cette sérialisation utilise l'hypothèse qu'un processus qui a reçu sa demande l'utilise puis restitue toutes les ressources au bout d'un temps fini. Si l'allocateur n'arrive pas à satisfaire tous les demandeurs en constituant une séquence, appelée suite saine, il n'y arrivera pas non plus en laissant des processus s'exécuter concurremment.

Si l'état est sain, on sait qu'il existe une voie de secours, c'est de servir les processus selon la suite saine trouvée. S'il n'existe aucune suite saine, alors le système est en interblocage.

On a aussi un résultat très intéressant. Supposons que le système soit dans un état sain et que l'allocateur puisse le faire passer dans un nouvel état par suite d'une demande d'un processus Pk. Pour déterminer si ce nouvel état est sain, on doit essayer de déterminer une séquence fictive et de construire une suite saine. Dès que le processus Pk est placé dans la séquence, on sait qu'une suite saine existe et que l'état est sain. Tout algorithme de recherche d'état sain peut se terminer quand Pk a été placé dans la séquence.

4.2. Applications de l'état sain

Toute politique d'allocation globale des ressources place toujours le système dans des états sains. Tout demandeur est soit servi soit bloqué sans ressources. Une suite saine s'obtient en plaçant dans la séquence fictive, d'abord tous les processus servis, puis tous les processus bloqués. Quand tous les processus servis auront rendus leurs ressources, l'allocateur aura récupéré toutes les ressources de l'état initial et peut servir un à un tous les autres processus. Il n'y a jamais interblocage.

Toute politique d'allocation des ressources par classes ordonnées place toujours le système dans des états sains. Tout demandeur est soit servi complètement, soit il lui manque les ressources de la classe m, soit il lui manque les ressources des classes m et m-1, etc., soit il est bloqué sans ressources. Une suite saine s'obtient en plaçant dans la séquence fictive, d'abord tous les processus servis, puis tous les processus en attente des ressources de la classe m, puis tous les processus en attente des ressources des classe m et m-1, etc., puis en dernier tous les processus bloqués. Quand tous les processus servis auront rendus leurs ressources, l'allocateur aura récupéré toutes les ressources de la classe m, il peut alors servir un à un tous les processus en attente des ressources de la classe m. Quand tous ces processus se seront exécuté et auront rendus leurs ressources, l'allocateur aura récupéré toutes les ressources des classes m et m-1. Et ainsi de suite pour chaque classe. Il n'y a jamais interblocage.

5. État fiable

5.1. Recherche d'une suite fiable complète

On doit montrer que l'état d'allocation atteint à t et défini par A(t) ne présente aucun risque de conduire le système vers un interblocage si un ou tous les processus se mettent à demander toutes les ressources qu'ils ont contractuellement annoncées et indiquées dans C.

Supposons que ce soit le cas. On examine si l'allocateur pourrait répondre à cette demande maximale, en faisant une exécution fictive séquentielle des processus et en supposant qu'un processus qui a reçu son annonce rend toutes les ressources au bout d'un temps fini. On examine pour cela la distance restant pour chaque processus et on doit pouvoir la servir à un moment de l'exécution séquentielle grâce aux ressources résiduelle et aux ressources rendues par les processus déjà terminés dans cette exécution. Cette exécution séquentielle prouve que l'état examiné est sans danger et que l'allocateur peut y placer le système.

Comme $Ai(t) \leq Di(t) \leq Ci \leq X$, un état fiable est aussi un état sain et un état sain est aussi un état réalisable.

Si l'état est fiable, on sait qu'il existe une voie de secours pour l'évitement, c'est de servir les processus selon la suite fiable trouvée. S'il n'existe aucune suite fiable, alors le système peut évoluer vers un interblocage.

On a aussi un résultat très intéressant. Supposons que le système est dans un état fiable et qu'il doit passer dans un nouvel état par suite d'une demande d'un processus Pk. Pour déterminer si ce nouvel état est fiable, on doit essayer de déterminer une séquence fictive et de construire une suite fiable. Dès que le processus Pk est placé dans la séquence, on sait qu'une suite fiable existe et que l'état est fiable. Tout algorithme de recherche d'état fiable peut se terminer quand Pk a été placé dans la séquence.

5.2. Applications de l'état fiable

L'état initial du système est fiable. On conduit le système en l'obligeant à rester dans des états fiables. L'allocateur de ressource n'accepte de servir une requête d'un processus Pk que si l'allocation correspondante met le système dans un état fiable. Si le résidu permet de servir la requête du processus Pk, l'allocateur, avant de faire l'allocation, considère le nouvel état qui résulterait de cette allocation et il essaie de construire pour cet état une sous-suite fiable qui contienne Pk. Si c'est possible, il sert la requête de Pk sinon, par prévention, il met Pk dans une file d'attente, bien qu'il ait de quoi servir la requête. A la libération de ressources, il essaie de servir, l'une après l'autre, les requêtes en attente, à condition que la satisfaction de chaque requête mette le système dans un nouvel état fiable. On veillera à examiner d'autres requêtes que la première de la file si celle-ci ne peut être servie car un service à l'ancienneté peut créer un interblocage. L'examen du nouvel état qui résulterait de l'allocation faite à Pk est réalisé par un algorithme de prévention appelé l'algorithme du banquier.

6. Algorithmes de prévention dynamique par report de service

6.1. Algorithme du banquier pour m classes de ressources

L'algorithme du banquier, conçu par E.W. Dijkstra et N. Habermann, reçoit en données les matrices C et A d'annonce et d'allocation, le vecteur R résidu de ressources, et k l'identité du processus demandeur. La matrice d'allocation comprend l'allocation demandée par Pk. Le résultat donne ou non l'accord du banquier en indiquant si l'état est fiable ou non.

L'algorithme construit une sous-suite fiable S à partir de l'ensemble T des processus non encore retenus pour S. A chaque itération, on essaie d'agrandir la sous-suite fiable S en y ajoutant un élément de T. Cet ajout n'est possible que si le processus candidat peut recevoir le reste de son annonce avec les ressources récupérées par l'allocateur dans le fonctionnement séquentiel fictif des processus qui font partie de la sous suite fiable. Ceci s'exprime en comparant Distance, l'écart par rapport à l'annonce, et Total, la valeur atteinte pour les ressources récupérées. Si le candidat n'est pas acceptable, on en essaie un autre dans T. S'il n'y a plus de candidat dans T, on a testé tous les processus et on ne peut pas garantir qu'il n'y aura jamais d'interblocage. l'algorithme répond faux. Si le candidat convient, on l'ajoute à S, on simule le résultat de son exécution fictive et le retour des ressources qu'il utilise en augmentant Total. On reconstitue l'ensemble T des candidats, ce sont tous les processus qui ne sont pas dans la sous-suite S.

On sait que le système est parti d'un état fiable et que le nouvel état n'en diffère que par la demande de Pk. On sait aussi qu'il suffit de placer Pk dans la sous-suite fiable S pour que le

nouvel état soit fiable. On utilise ces résultats pour simplifier l'algorithme et essayer de placer Pk dans S dès que possible. On arrête alors l'itération. Un test après la sortie permet de savoir si l'arrêt de l'itération est dû au placement de Pk ou non, et si l'état est fiable ou non.

La complexité de l'algorithme s'exprime en nombre d'itérations. La situation la plus complexe est celle où la demande de Pk ne peut être satisfaite qu'en dernier, grâce au retour des ressources de tous les autres processus et où le processus qu'on ajoute à la sous-suite à chaque étape est le dernier de T. Le premier de la sous-suite est trouvé après N itérations, le second après N-1 itérations,..., l'avant dernier en deux itérations et le dernier, Pk, en une itération. Cela donne n(n+1)/2 itérations et la complexité est en $O(n^2)$

Un traitement heuristique, introduit par R. Holt, permet de ramener la complexité à O(m*n*logn) en améliorant la recherche du prochain candidat pour la sous-suite S. On crée m listes, une par classe i de ressource et, dans chaque liste, on trie les processus Pj selon Dist(i, j) et on marque ceux pour qui Dist(i, j) ≤ Total(i). On choisit comme candidat un processus Pj marqué dans toutes les listes. A chaque itération, Total(i) augmente et on marque de nouveaux processus dans chaque liste i.

6.2. Algorithme du banquier pour une seule classe de ressources

Avec une seule classe de ressources, l'heuristique devient simple à mettre en oeuvre. Les processus sont triés par Dist croissant et à chaque itération, on ne fait que deux tests, l'un sur Pk, l'autre sur le premier processus de T ainsi trié. Il y a échec si le second test est négatif.

Pour implanter l'algorithme, on suppose que le tri des processus selon Dist(i) est indiqué dans Rang. Alors Rang(1) donne l'indice du processus qui sort premier du tri et Rang(j) donne celui qui est classé au rang j. A chaque itération j, les deux tests concernent le processus Pk et le processus de Rang(j).

6.3. Prévention avec des annonces égales pour une classe de ressources

6.3.1. *Simplifications*

Si toutes les annonces sont égales et valent C, on fait au plus une itération, car dès que le processus le mieux servi, Rang(1), appelé le Leader, libère son annonce, tous les autres peuvent se terminer. Donc le nouvel état t' est fiable si, à t' :
(Leader' = k) et (Dist'(k) ≤ R')
ou si (Leader' ≠ k) et (Dist'(Leader) ≤ R') et (Dist'(k) ≤ A'(Rang(1)) + R') .
Considérons l'état t avant allocation (état fiable par hypothèse) et ramenons la condition à cet instant en utilisant req(k) la requête du processus k à t . Il vient :

• l'état est fiable à t :	Dist(Leader) ≤ R
• pour que l'allocateur puisse servir Pk :	req(k) ≤ R
• cas où Pk devient leader à t':	Dist(k) - req(k) ≤ Dist(Leader)
il faut aussi :	Dist(k) - req(k) ≤ R - req(k) donc Dist(k) ≤ R
• cas où Pk ne devient pas leader :	Dist(Leader) ≤ Dist(k) - req(k)
l'état doit être fiable, donc :	Dist(Leader) ≤ R - req(k)
et aussi :	Dist(k) - req(k) ≤ A(Leader) + R - req(k)

Cette dernière relation se déduit de ce que toutes les annonces valent C.

On a Dist(k) ≤ C et puis C = A(Leader) + Dist(Leader) et puis Dist(Leader) ≤ R - req(k), cela entraîne que Dist(k) ≤ A(Leader) + R - req(k) et a fortiori Dist(k) ≤ A(Leader) + R

6.3.2. *Algorithme de prévention pour des annonces égales*

Le test de prévention consiste donc à garder de quoi satisfaire l'annonce du leader après allocation, qu'il ait changé ou non à la suite de cette allocation. Il vient :

```
function EtatFiable return boolean is
      OK : Boolean := false;
begin
    if req(k) <= R then
        if (req(k) >= Dist(k) - Dist(Leader)) then -- c'est Pk le nouveau leader potentiel
                OK := (Dist(k) ≤ R);
                if OK then Leader := k; end if ; -- on confirme Pk en leader
        else -- on a  (req(k) < Dist(k) - Dist(Leader)) et Pk n'est pas le nouveau leader
                OK := (Dist(Leader) ≤ R - req(k));
        end if;
    else
        OK := False; -- il n'y a pas de ressources pour satisfaire la requête
    end if;
    return OK;
end EtatFiable;
```

6.3.3. *Autre algorithme de prévention pour des annonces égales*

On ne cherche pas à identifier le leader, mais on se contente de vérifier que, dans le nouvel état après allocation, un processus au moins pourra atteindre son annonce maximale.

```
    -- le processus k demande req(k) ressources en plus et sa distance est Dist(k)
function EtatFiable(k : in IdProc) return Boolean is
    OK : Boolean;  -- après allocation, on peut encore servir l'annonce d'au moins un processus
begin
    if req(k) > R then
        return False ;
    else
        R := R - req(k) ;  -- on attribue les req(k) ressources à k, pour voir
        Dist(k) := Dist(k) - req(k) ;
        for J in 1..IdProc loop
                OK := (Dist(J) <= R); -- un processus au moins pourra atteindre son annonce
                exit when OK; -- on en a trouvé 1, c'est bon
        end loop;
        R := R + req(k) ;  -- on retourne à l'état initial
        Dist(k) := Dist(k) + req(k) ;
        return OK;
    end if;
end EtatFiable;
```

6.4. Politique de précaution permanente avec une classe de ressources

On donne au système assez de ressources pour que même dans le pire cas, tous les processus ne puissent jamais être bloqués simultanément. Le test du banquier est superflu car il est toujours vrai. Pour que tous les processus puissent être bloqués en même temps, il faudrait que $\sum(Ci - 1) = X$. Dès que $X = q + \sum(Ci - 1)$, on s'assure que q processus ne sont jamais bloqués.

Si toutes les annonces sont égales à C, la précaution demande que $n*C < X + n$.

QUELQUES EXERCICES POUR LE CHAPITRE 06

Exercice 1 (mai 1991). Un système est en interblocage quand:

a-les processus les moins prioritaires ne sont jamais activés,

b-les cases de mémoire attribuées précédemment aux processus prêts sont reprises par l'algorithme de remplacement pour charger des pages du processus élu,

c-des processus bloqués s'attendent circulairement,

d-les interruptions sont masquées et des processus sont en attente active.

Exercice 2 (juin 1991)

On dispose d'un certain nombre de ressources critiques nécessaires au fonctionnement de programmes telles que imprimante graphique, imprimante couleur, table traçante, modem,...

Chaque programme, lorsqu'il s'exécute, demande l'allocation des ressources qui lui sont nécessaires. Lorsqu'il a fini de s'exécuter, il libère ces ressources. Cela donne en particulier le cas suivant:

```
task body P1 is                task body P2 is                task body P3 is
begin                          begin                          begin
demander(table_traçante);      demander(modem);               demander(imprimante);
demander(modem);               demander(imprimante);          demander(table_traçante);
...exécution...                ...exécution...                ...exécution...
libérer(modem);                libérer(imprimante);           libérer(table_traçante);
libérer(table_traçante);       libérer(modem);                libérer(imprimante);
end P1;                        end P2;                        end P3;
```

Question 1. Cette solution n'est pas satisfaisante dans tous les cas; expliquer pourquoi.
Question 2. Proposer une solution ne posant plus de problème.

Exercice 3 (juin 1993)**.** Quatre processus P1, P2, P3 et P4 se partagent 50 granules sur un disque. Chacun garde les granules acquis et attend l'arrivée de sa nouvelle demande avant de continuer. Il n'y a pas de réquisition.

P1 demande successivement 1 granule, puis 4, puis 3, puis 6, puis 7, puis 1, puis rend les 22 granules acquis.

P2 demande successivement 4 granules, puis 3, puis 9, puis 5, puis 5, puis rend les 26 granules acquis.

P3 demande successivement 2 granules, puis 3, puis 4, puis 7, puis 2, puis 2, puis 1, puis rend les 21 granules acquis.

P4 demande successivement 7 granules, puis 2, puis 1, puis 2, puis 2, puis 9, puis rend les 23 granules acquis.

Question 1) :Montrer que l'état suivant correspond à un interblocage du système :

R(t1) = 6
A(t1) = (14, 7, 9, 14)
D(t1) = (21, 16, 16, 23)

Donner un autre exemple d'état d'interblocage du système.

Question 2) : On prend la valeur de 26 granules comme annonce maximale pour chacun des processus du système. En garantissant au processus le mieux pourvu en granules la possibilité d'atteindre cette valeur maximale, on peut simplifier la politique de prévention dynamique de l'interblocage. Rappeler cette politique et expliquer pourquoi cette simplification est valable.

En particulier, quand il applique cette politique(simplifiée), que doit répondre l'allocateur s'il est confronté dans l'état t3 aux demandes D(t3) :

R(t3) = 18
A(t3) = (14, 7, 3, 8)
D(t3) = (14, 16, 9, 16)

C - A(t3) = (12, 19, 23, 18) ?

L'allocateur peut-il satisfaire n'importe lequel des processus demandeurs (autre que P1) en étant assuré de ne jamais laisser le système aller vers un état d'interblocage? Expliquer votre réponse pour chaque processus.

Exercice 4 (septembre 90)**.** Soit un système qui gère un total de 48K de mémoire allouable, qui sert les requêtes des processus sans réquisition et qui bloque tout processus demandeur tant que sa requête ne peut être satisfaite. Trois processus PA, PB, Pc ont déclaré au préalable que leur demande maximale serait de 25K, 15K et 41K respectivement. A un moment de l'exécution, les 3 processus sont actifs tous les 3 et ont acquis 3K, 9K et 24K respectivement. Laquelle (ou lesquelles) des requêtes suivantes de mémoire supplémentaire peut elle être servie avec l'assurance qu'il n'en résultera jamais d'interblocage, une fois l'allocation faite:

a- PA demande 9K,
b- PC demande 7K,
c- PB demande 6K,
d- PA demande 6K.

Exercice 5 (février 1997)

Question 1) 5 processus se partagent 11 ressources d'une classe de ressources banalisées qu'ils demandent une à une (par exemple des granules de mémoire secondaire). Chaque processus demande au total au plus 3 ressources. Montrer qu'il ne peut y avoir interblocage. (piste : il y aurait interblocage si chaque processus avait reçu x ressources et en demandait encore plus et s'il n'y a plus assez de ressources pour permettre à un processus au moins d'obtenir trois ressources).

Question 2) On généralise cet exemple : N processus se partagent M ressources d'une classe de ressources banalisées qu'ils demandent une à une. La demande totale T de chaque processus ne peut pas dépasser M ressources et la somme des demandes totales de tous les processus N * T est plus petite que M + N. Montrer qu'il ne peut pas y avoir interblocage.

Question 3) Si 5 processus se partagent 6 ressources d'une classe de ressources banalisées qu'ils demandent une à une (par exemple des segments de mémoire principale) et si chaque processus demande au total au plus 2 ressources, il ne peut y avoir interblocage selon le résultat du 2 ci-dessus. Supposons maintenant que 5 processus se partagent 6 ressources d'une classe A de ressources banalisées qu'ils demandent une à une sans dépasser le total Ta = 2 et aussi 11 ressources d'une classe B de ressources banalisées qu'ils demandent une à une sans dépasser le total Tb = 3. Montrer que contrairement au cas où il n'y a qu'une classe de ressources, la présence de deux classes de ressources peut conduire à interblocage si chaque processus peut demander les ressources une à une dans une classe ou l'autre, sans dépasser les totaux Ta de A et Tb de B.

Question 4) Montrer qu'avec 7 ressources de classe A et 14 ressources de classe B, il n'y a pas d'interblocage; (pour vous mettre sur la piste de la solution : il en sera de même, et il n'y a pas interblocage, avec 6 ressources de classe A et 15 ressources de classe B; ou avec 8 ressources de classe A et 13 ressources de classe B; ou ...; ou avec 10 ressources de classe A et 11 ressources de classe B).

RÉPONSES AUX EXERCICES

Réponse à l'exercice 1 : c

Réponses à l'exercice 2

1) attente circulaire possible

2) P3 doit demander la table traçante avant l'imprimante

Réponses à l'exercice 3

1) il y a interblocage car R(t1) vaut 6 ce qui est inférieur à chacune des demandes des processus
autres exemples d'interblocage (parmi d'autres)

A( ) = (14, 21, 2, 12) D( ) = (21, 26, 5, 14) R( ) = 1
A( ) = (14, 16, 5, 14) D( ) = (21, 21, 9, 23) R( ) = 1
A( ) = (0, 21, 20, 9) D( ) = (1, 26, 21, 10) R( ) = 0
A( ) = (8, 16, 9, 14) D( ) = (14, 21, 16, 23) R( ) = 3
A( ) = (5, 21, 9, 14) D( ) = (8, 26, 16, 23) R( ) = 1

2) La solution consiste à appliquer l'algorithme du banquier qui se simplifie ici : il suffit de toujours garder à t, après allocation fiable ou refus, de quoi servir la demande maximale du processus qui a obtenu à t le plus de ressources (c'est aussi celui qui a la plus petite valeur pour C - Ai).

Donc pour déterminer les états fiables à partir de t3, on essaie de servir 1 processus et on applique la règle.

R(t3) = 18
A(t3) = (14, 7, 3, 8)
D(t3) = (14, 16, 9, 16)

C - A(t3) = (12, 19, 23, 18) ?

Servir P2 donnerait, (en lui allouant 9 granules)

A(t4) = (14, 16, 3, 8)
D(t4) = (14, 16, 9, 16)

C - A(t4) = (12, 10, 23, 18) R(t4) = 9 Aucun processus ne peut atteindre 26. On ne doit pas servir P2

Servir P3 donnerait, (en lui allouant 6 granules)

A(t4) = (14, 7, 9, 8)
D(t4) = (14, 16, 9, 16)

C - A(t4) = (12, 19, 17, 18) R(t4) = 12 P1 peut atteindre 26. On peut servir P3

Servir P4 donnerait , (en lui allouant 8 granules)

A(t4) = (14, 7, 3, 16)
D(t4) = (14, 16, 9, 16)

C - A(t4) = (12, 19, 23, 10) R(t4) = 10 P4 peut atteindre 26. On peut servir P4

Réponse à l'exercice 4 : A5.c

Réponses à l'exercice 5

1) au pire, chaque processus a obtenu déjà 2 ressources, ce qui a mobilisé 10 ressources. Le premier qui demandera sa troisième ressource pourra terminer et rendre une ressource au moins; cette ressource pourra permettre à un autre processus de terminer, ...

2) au pire, les N processus ont chacun T - 1 ressources; il doit en rester une pour que l'un après l'autre tous les processus puissent terminer, donc M = N(T - 1) + 1 ou encore
M + N = N * T + 1 ou encore M + N > N * T

3) soit le tableau donnant les allocations avec Ta = 2 et Tb = 3

processus	P1	P2	P3	P4	P5	reserve
ressource A	1	1	1	1	1	1
ressource B	2	2	2	2	2	1

Si P1 reçoit la dernière ressource A et P2 la dernière ressource B, on est en interblocage dès que tous les processus demandent une nouvelle ressource qu'elle soit de la classe A ou de la classe B.

4) Pour éviter l'interblocage, il faut que, dans le pire cas, la dernière ressource puisse permettre aux processus de finir l'un après l'autre. Il vient :

processus	P1	P2	P3	P4	P5	reserve
ressource A	2	2	1	1	1	0
ressource B	2	2	3	3	3	1