Interblocages – Sujet CorrigÃl'

1 Caractérisation

Question. Rappelez les quatre conditions qui peuvent conduire à un interblocage. Est-il nécessaire qu'elles soient toutes vérifiées pour qu'un interblocage puisse se produire ? Sont-elles indépendantes ?

Question. Rappelez la définition du graphe ressource-allocation.

Soit un système ayant 3 processus P_1, P_2, P_3, 4 types de ressources R_1, R_2, R_3, R_4, de nombre d'instances respectives 1, 2, 1, 3, se trouvant dans l'état suivant dans l'état suivant :
- P_1 détient une instance de type R_2 et attend une instance de type R_1 ;
- P_2 détient une instance de type R_1 et de type R_2 et attend une instance de type R_3 ;
- P_3 détient une instance de type R_3

Question. Quel est le graphe ressource-allocation associé ? Dessinez-le.

Question. Y a-t-il un risque d'interblocage ? Sur quelle propriété vous basez-vous ? Est-ce une condition nécessaire et suffisante ? Si oui, prouvez-le. Si non, donnez un contre-exemple et expliquez dans quelle cas cette propriété devient une condition nécessaire et suffisante. Que se passe-t-il si P_3 demande une instance de type R_2 ?

2 Prévention

Question. Proposez ou rappelez un moyen d'empêcher la réalisation de chacune des conditions pouvant amener à un interblocage. Jugez la pertinence de ces propositions. Une attention particulière sera portée sur la *condition d'attente circulaire*. En quoi ces mécanismes des prévention sont-ils contraignants ?

Correction

Mutual Exclusion not required for sharable resources ; must hold for nonsharable resources.

Hold and Wait must guarantee that whenever a process requests a resource, it does not hold any other resources.
- Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
- Low resource utilization ; starvation possible.

No Preemption – If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
- Preempted resources are added to the list of resources for which the process is waiting.
- Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Circular Wait impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

□

3 Évitement

Un mécanisme d'évitement utilise de l'information supplémentaire pour éviter de rendre les interblocages possibles. Le modèle le plus simple et probablement le plus utile requiert que chaque processus déclare la quantité maximale de chaque type de ressources dont il peut avoir besoin. Nous nous basons sur ce modèle et définissons alors un état *sain* comme il suit :

Définition. `État sain ou aventureux.` *Un état est* sain *si le système peut allouer les ressources à chaque processus (jusqu'à son maximum) dans un certain ordre et finalement éviter un interblocage. Un état qui n'est pas sain sera qualifié d'état* aventureux*.*

Question. Quelles sont les relations entre états saufs, aventureux et les interblocages ?

On suppose pour l'instant qu'il n'existe qu'une seule instance de chaque ressource.

Question. Proposez un algorithme d'évitement d'interblocage basé sur le graphe ressource-allocation. Vous pourrez introduire un nouveau type d'arête $P_i \rightarrow R_j$ indiquant qu'un processus P_i peut demander R_j à un moment donné dans le futur. Illustrez votre algorithme sur un exemple comptant 2 processus et 2 ressources.

Correction

1. Claim edge $P_i \rightarrow R_j$ indicated that process P_j may request resource R_j ; represented by a dashed line.
2. Claim edge converts to request edge when a process requests a resource.
3. Request edge converted to an assignment edge when the resource is allocated to the process.
4. When a resource is released by a process, assignment edge reconverts to a claim edge.
5. Resources must be claimed a priori in the system.

□

On suppose maintenant que plusieurs instances de chaque type de ressource peuvent exister. Soient n le nombre de processus et m le nombre de types de ressources. Soient les quatre structures de données suivantes :

Dispo Un vecteur de longueur m indiquant le nombre disponible de chaque type de ressource à l'instant présent.

Max Une matrice $n * m$ indiquant la demande maximale de chaque processus sur chaque ressource (que chaque processus donne à l'entrée dans le système).

Alloc Une matrice $n * m$ indiquant le nombre courant de chaque type de ressource alloué à chaque processus.

Besoin Une matrice $n * m$ indiquant la quantité d'instances de chaque type de ressource dont *peut* avoir *de surcroît* besoin chaque processus. On a : $Besoin = Max - Alloc$.

Question. Proposez un algorithme permettant de décider si un système est dans un état *sain*. Quelle est sa complexité ?

Correction Algorithme de Banker.

1. Let Work and Finish be vectors of length m and n, respectively. Initialize :
   $\mathrm{Work} = \mathrm{Available}$
   $\mathrm{Finish}[i] = \mathrm{FALSE}$ for $i = 0, 1, \ldots, n-1$.
2. Find an i such that both :
   – $\mathrm{Finish}[i] = \mathrm{FALSE}$,
   – $\mathrm{Need}_i \leqslant \mathrm{Work}$.
   If no such i exists, go to step 4.
3. $\mathrm{Work} = \mathrm{Work} + \mathrm{Allocation}[i]$
   $\mathrm{Finish}[i] = \mathrm{TRUE}$
   go to step 2.
4. If $\mathrm{Finish}[i] == \mathrm{TRUE}$ for all i, then the system is in a safe state.

□

Question. Déduisez-en un algorithme permettant de décider si une requête d'allocation peut-être satisfaite de manière sauve. Quelle analogie peut-on faire avec le problème d'un banquier proposant davantage de crédit qu'il ne possède de liquidité ?

Correction $\mathrm{Request}$ = request vector for process P_i. If $\mathrm{Request}_i[j] = k$ then process P_i wants k instances of resource type R_j.

1. If $\mathrm{Request}_i \leqslant \mathrm{Need}_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $\mathrm{Request}_i \leqslant \mathrm{Available}$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows :
   – $\mathrm{Available} = \mathrm{Available} - \mathrm{Request}$;
   – $\mathrm{Allocation}_i = \mathrm{Allocation}_i + \mathrm{Request}_i$;
   – $\mathrm{Need}_i = \mathrm{Need}_i - \mathrm{Request}_i$;

   If safe $\Rightarrow$ the resources are allocated to P_i.
   If unsafe $\Rightarrow$ P_i must wait, and the old resource- allocation state is restored

□

4 Détection

Si aucun mécanisme de prévention ou d'évitement n'est utilisé, un interblocage peut avoir lieu. Dans ce cas, un système peut fournir un mécanisme basé sur la détection et le recouvrement. Nous nous intéressons ici à la partie détection.

On suppose pour l'instant qu'il n'existe qu'une seule instance de chaque ressource.

Question. Comment déterminer un interblocage à partir du graphe ressource-allocation. En déduire un mécanisme possible de détection. Quel est son coût ?

Correction

- Maintain wait-forgraph
  - Nodes are processes.
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j.
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

□

On suppose désormais que plusieurs instances de chaque type de ressource peuvent exister. Soient n le nombre de processus et m le nombre de types de ressources. Soient les quatre structures de données suivantes :

Dispo Un vecteur de longueur m indiquant le nombre disponible de chaque type de ressource à l'instant présent.

Alloc Une matrice $n * m$ indiquant le nombre courant de chaque type de ressource alloué à chaque processus.

Requete Une matrice $n * m$ indiquant le nombre courant de requêtes supplémentaires de chaque processus sur chaque ressource.

Question. En vous inspirant de l'algorithme de la section précédente, proposez un algorithme de détection d'interblocage. Quelle est sa complexité ?

Correction

1. Let Work and Finish be vectors of length m and n, respectively.
   Initialize :
   (a) $\mathrm{Work} = \mathrm{Available}$.
   (b) For $i = 1, 2, \ldots, n$, if $\mathrm{Allocation}_i \neq 0$, then $\mathrm{Finish}[i] = \mathrm{FALSE}$;otherwise, $\mathrm{Finish}[i] = \mathrm{TRUE}$.
2. Find an index i such that both :
   (a) $\mathrm{Finish}[i] == \mathrm{FALSE}$,
   (b) $\mathrm{Request}_i \leqslant \mathrm{Work}$.
   If no such i exists, go to step 4.
3. $\mathrm{Work} = \mathrm{Work} + \mathrm{Allocation}_i$
   $\mathrm{Finish}[i] = \mathrm{TRUE}$
   Go to step 2.
4. If $\mathrm{Finish}[i] == \mathrm{FALSE}$, for some i, $1 \leqslant i \leqslant n$, then the system is in deadlock state. Moreover, if $\mathrm{Finish}[i] == \mathrm{FALSE}$, then P_i is deadlocked.

Algorithm requires an order of $O\left(mn^2\right)$ operations to detect whether the system is in deadlocked state. □